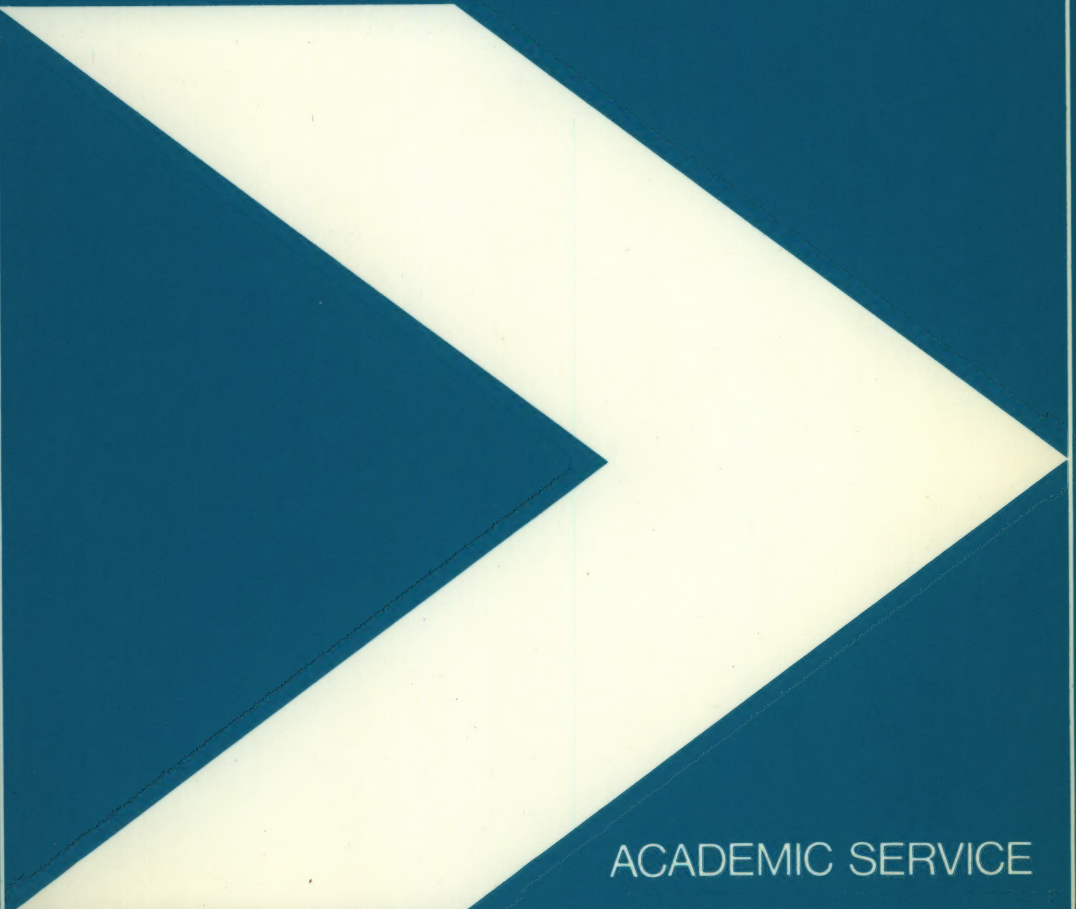


bestands- organisatie

prof. dr. R.J. LUNBECK en drs. F. REMMEN



ACADEMIC SERVICE



BESTANDSORGANISATIE

Andere uitgaven van prof. dr. R. J. Lunbeck

- 'Inleiding Programmeren', Academic Service, Den Haag
- 'Inleiding Datastructuren', Academic Service, Den Haag

Andere uitgaven van Drs. F. Remmen

- 'Inleiding in de informatica', Stenfert Kroese, Leiden
- 'COBOL', Stenfert Kroese, Leiden

bestands- organisatie

prof. dr. R.J. LUNBECK en drs. F. REMMEN

ACADEMIC SERVICE

CIP-GEGEVENS

Lunbeck, R.J.

Bestandsorganisatie / R.J. Lunbeck en F. Remmen. - Den Haag :
Academic Service. - Ill.

Met lit. opg., reg.

ISBN 90-6233-075-4

SISO 365.1 UDC 681.3.016

Trefw. : bestandsorganisatie.

1e druk 1975

2e herziene uitgave 1977

3e bijdruk 1979

4e herziene uitgave 1980

5e gewijzigde druk 1982

6e ongewijzigde druk 1984

7e bijdruk 1985

8e druk 1986

Uitgegeven door: Academic Service

Postbus 96996

2509 JJ Den Haag

Druk: Krips Repro Meppel

Bindwerk: Meeuwis, Amsterdam

Omslagontwerp: JAM Gauw

ISBN 90 6233 075 4

© 1975 Academic Service, auteursrechten voorbehouden

Niets uit deze uitgave mag worden verveelvoudigd en/of openbaar gemaakt door middel van druk, fotokopie, microfilm, geluidsband, elektronisch of op welke andere wijze ook, en evenmin in een retrieval system worden opgeslagen zonder voorafgaande schriftelijke toestemming van de uitgever.

VOORWOORD BIJ DE VIERDE DRUK

Zoals de titel aangeeft is het de bedoeling de lezer een gedegen inzicht te verschaffen in wat wel de klassieke bestandsorganisatie wordt genoemd. De opzet is echter zodanig dat het boek tevens gebruikt kan worden als inleiding voor de database-methoden van de laatste jaren. Met name het laatste hoofdstuk geeft een inleiding tot en een overzicht van deze methoden.

Na een enigszins gewijzigde tweede druk en een ongewijzigde derde druk is, mede in overleg met andere docenten, besloten het boek wat ingrijpender te herzien. Deze herziening raakt echter niet de hoofdopzet, nl. naast een statische beschrijving van bestandsstructuren vooral ook programmaschetsen te geven van het werken met deze structuren. De meest opvallende veranderingen betreffen:

- een uitvoeriger behandeling met programmaschetsen van de directe en index-sequentiële bestandsorganisatie in plaats van de meer beschrijvende behandeling in vroegere drukken,
- het gebruik van B-bomen voor een recente mengvorm van de sequentiële en de directe bestandsorganisatie,
- een uitvoeriger behandeling van geïnverteerde bestanden,
- een gedeeltelijke herrangschikking van de stof en het weglaten van enkele onderwerpen (quicksort, database vraagtafen),
- het gebruik van de in de nieuwste literatuur gebezigde termen.

Aan iedereen die ons zijn reactie stuurde naar aanleiding van vroegere drukken, willen wij onze erkentelijkheid betuigen. Wij houden ons wederom aanbevolen voor op- en aanmerkingen die ongetwijfeld los zullen komen, omdat wij door omstandigheden wederom in tijdnood kwamen bij het klaarmaken van deze druk. Het is slechts aan de uitgever en zijn medewerkers te danken dat dit boek nog op tijd voor de nieuwe cursus zal uitkomen.

Eindhoven, juli 1980

R. J. Lunbeck
F. Remmen

VOORWOORD BIJ DE VIJFDE DRUK

Toch weer eerder dan verwacht werden wij er op geattendeerd dat de vierde druk uitverkocht raakte.

Dank zij de opmerkingen van vele attente lezers, konden in deze nieuwe druk verschillende, soms enigszins storende fouten uit de vierde druk worden verbeterd.

Verder is er een nieuw hoofdstuk: "Keuze van bestandsorganisatie" aan toegevoegd. Hiermee is, menen wij, tegemoet gekomen aan een wens van vele gebruikers, die na de behandeling van verschillende vormen van bestandsorganisatie de behoefte gevoelen daarna expliciet het keuzeprobleem naar voren te brengen.

Wij houden ons uiteraard weer gaarne aanbevolen voor op- en aanmerkingen van de gebruikers van dit leerboek.

Eindhoven, februari 1982

R. J. Lunbeck
F. Remmen

INHOUD

1.	INFORMATIESYSTEMEN	9
1.1	Inleiding	9
1.2	Informatiesysteem. Informatiebehoefte	11
1.3	Achtergrondgeheugens (backing store)	14
2.	BASISBEGRIPPEN BIJ GEGEVENSSTRUKTUREN	21
2.1	Objekten en attributen; logische structuur	21
2.2	Typen en individuen	24
2.3	Opslagstructuren	26
	2.3.1 Opslagstructuur voor een record	27
	2.3.2 Opslagstructuur voor een bestand. Algemene inleiding	30
3.	SEQUENTIELE BESTANDSORGANISATIE	36
3.1	Inleiding	36
3.2	Onderhoud; bestandsmutaties	37
3.3	Sequentieel georganiseerd bestand op een adresseer- baar medium	48
3.4	Aparte problemen bij verwerking	50
4.	LIJSTSTRUKTUREN	57
4.1	Inleiding	57
4.2	Soorten lijststructuren	60
4.3	Combinatie van consecutieve structuur en lijststruk- turen	66
4.4	Voorbeelden van verwerking op basis van een gegeven lijststructuur	66

5.	SORTEREN EN ZOEKEN	76
5.1	Inleiding	76
5.2	Sorteren	76
5.2.1	Interne sortering	77
5.2.2	Externe sortering	81
5.3	Zoekmethoden	85
6.	DIRECTE BESTANDSORGANISATIE	90
6.1	Inleiding	90
6.2	Opslagstructuur voor directe bestandsorganisatie	91
6.3	Sleutelconversie bij funktionele relatie	98
6.4	Overflow bij een funktionele relatie	103
6.5	Overloop in vrije lokaties (scatter storage)	107
6.6	Onderhoud en gebruik van een direct georganiseerd bestand	109
7.	COMBINATIES VAN GRONDVORMEN VAN BESTANDS-ORGANISATIE	120
7.1	Index-sequentiële bestandsorganisatie met tabellen	120
7.1.1	Inleiding	120
7.1.2	Structuur voor sequentiële en directe toegang	121
7.1.3	Consequenties van bestandsmutaties	124
7.1.4	Enkele varianten van de I-S-organisatie	124
7.1.5	Programmaschetsen voor direct zoeken in een I-S-bestand	129
7.1.6	Onderhoud van een I-S-bestand	133
7.2	Directe bestandsorganisatie voor een bestand met grote recordlengte	135
7.3	Een andere tussenvorm van de sequentiële en directe organisatie met behulp van bomen	137
7.3.1	Inleiding	137
7.3.2	Zoeken van een record	137
7.3.3	Toevoegen van een record	138
7.3.4	Verwijderen van een record	139
7.3.5	B ⁺ -bomen voor sequentiële en directe toegang	140
8.	GEINVERTEERDE BESTANDEN	144
8.1	Geïnverteerd bestand als anders gesorteerd bestand	144
8.2	Geïnverteerd bestand als verzameling van tabellen	145
8.3	Uitgewerkt voorbeeld	147
8.4	Gebruik van geïnverteerde bestanden	149
8.5	Thesauri	149

9.	KEUZE VAN BESTANDSORGANISATIE	151
10.	DATABASES	157
10.1	Inleiding	157
10.2	De logische structuur van een database. Normalisatie	162
10.3	Database modellen	167
10.4	Gebruik van databases. Vraagstukken	175
11.	GEMENGDE OPGAVEN	178
APPENDIX I	EIGENSCHAPPEN VAN ACHTERGROND- GEHEUGENS	190
	1. Magneetband	190
	2. Magneettrommel	195
	3. Magneetschijven	197
	4. Magneetstripgeheugens	198
	5. Informatietransport tussen secundaire informatiedragers en hoofdgeheugen	199
APPENDIX II	TOELICHTING OP DE TAAL GEBRUIKT IN PROGRAMMASCHETSEN	203
	LITERATUURLIJST	206
	REGISTER	208

1 INFORMATIESYSTEMEN

1.1 INLEIDING

In dit hoofdstuk zal worden aangegeven, waar de bestandsorganisatie een belangrijke rol speelt. Bestandsorganisatie moet nl. niet als een doel op zichzelf worden beschouwd; het krijgt pas zijn volle betekenis als het wordt gezien als een onmisbaar hulpmiddel voor het goed functioneren van een informatiesysteem.

Aangezien bestandsorganisatie een afgeleide is van (de specificaties van) een informatiesysteem, zullen we eerst enige aandacht besteden aan het begrip informatiesysteem en de daarmee samenhangende begrippen informatie en informatiebehoefte. Vervolgens wordt kort ingegaan op 'achtergrondgegevens', waarop 'permanente' gegevens worden vastgelegd.

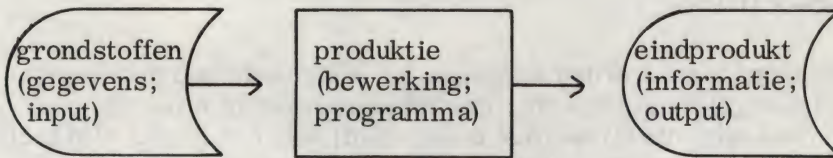
De hedendaagse maatschappij zou in zijn huidige vorm nauwelijks meer kunnen functioneren wanneer niet op alle plaatsen, waar over het wel en wee van deze maatschappij wordt beschikt, de nodige informatie voorhanden zou zijn. Deze informatie moet niet alleen juist zijn, maar ook zo tijdig en snel ter beschikking komen dat inderdaad effectieve beslissingen genomen kunnen worden.

Wat is eigenlijk informatie? Volgens een woordenboek is het een ander woord voor inlichtingen, mededelingen of berichten en als zodanig zullen we dit woord ook gebruiken. Ten onrechte wordt informatie soms gebruikt als synoniem voor gegevens ('data'); immers gegevens geven slechts in een bepaalde context geplaatst, informatie. Een gegeven dat de temperatuur 40°C is geeft in samenhang met koffie de informatie dat deze drank maar lauw is; in samenhang met woestijn de informatie dat de levensomstandigheden niet zo aangenaam zijn. Meneer X met een jaarinkomen van f 7000,-, student in Eindhoven, wordt beklaagd, doch meneer X met hetzelfde jaarinkomen in een ontwikkelingsland (gemiddeld jaarinkomen b.v. f 1000,-) komt op zijn minst voor nivellering in aanmerking.

Het vaak gebruikte woord 'informatieverwerking' is dan ook niet juist, want wat eigenlijk verwerkt wordt zijn 'gegevens'. Deze

verwerking vindt juist plaats in de hoop dat het resultaat voor een bepaalde situatie informatie zal bevatten.

Informatie kan worden beschouwd als een gevraagd eindprodukt terwijl gegevens grondstoffen zijn voor de aanmaak van het eindprodukt. Het maken van zo'n informatie-eindprodukt geschiedt door een bewerking van ter beschikking staande gegevens. Deze bewerkingen omvatten menselijke handelingen en het uitvoeren van computerprogramma's. De grondstoffen voor de bewerkingen – dus de gegevens – worden dikwijls weergegeven met de benaming input en het eindprodukt – dus de informatie – met de benaming output. Schematisch krijgen we dan het volgende beeld:



Bij dit schema is het volgende op te merken.

- a. Dikwijls zullen vele soorten grondstoffen nodig zijn voor een bewerking en/of verschillende soorten eindprodukten worden verlangd. Bijvoorbeeld:
 1. vele grondstoffen: gegevens over werknemers, gegevens over orders, gegevens over materialen, enz.
 2. verschillende produkten: overzicht gewerkte uren, facturen, bestellingen, enz.
- b. Om op efficiënte wijze goede informatie-eindprodukten te verkrijgen, zal het dikwijls nodig zijn gebruik te maken van 'half-fabrikaten', die ieder op zich weer het resultaat zijn van een bepaalde (tussen-)bewerking. Bijvoorbeeld:
om een jaaroverzicht te kunnen samenstellen, zal men gebruik maken van maandoverzichten, die zelf als output van een maandelijks verwerking ter beschikking zijn gekomen.

In produktiesystemen, waaraan bovenstaande voorbeelden zijn ontleend, wordt niet door het produkt zelf maar door de behoefte van de gebruiker bepaald of een produkt als halffabrikaat dan wel als eindprodukt dient te worden beschouwd. Men denke bijv. aan een industrie met een uitgebreid scala van produkten en een zeer uiteenlopende groep van afnemers.

Ditzelfde vinden we terug bij de produktie van informatie. Ook hier geldt dat de behoefte van de gebruikers bepaalt of sprake is van 'uiteindelijk gewenste informatie' of van 'tusseninformatie'.

Zo zal een afnemer alleen geïnteresseerd zijn in zijn eigen faktuurbedrag, terwijl voor de producent aparte faktuurbedragen

(een onderdeel van) de input vormen om de totale omzet te berekenen.

Een informatiesysteem kan dus worden beschouwd als een bijzonder soort produktiesysteem, nl. een systeem, dat informatie produceert.

1.2 INFORMATIESYSTEEM. INFORMATIEBEHOEFTE

Aangezien er behoefte is aan (velerlei soorten) informatie, wordt door de maatschappij gebruik gemaakt van vele informatiesystemen. In de literatuur kan men gevariëerde definities en omschrijvingen van dit begrip (informatiesysteem) vinden. Wij gebruiken hier de volgende omschrijving:

Een informatiesysteem is een geheel van methoden, faciliteiten en activiteiten, waarmee een organisatie haar informatiebehoeften zo goed mogelijk tracht te bevredigen.

methoden : behandelingswijzen, werkvoorschriften, enz.

faciliteiten : menselijke, materiële en financiële hulpmiddelen, enz.

activiteiten : (groeperingen van) werkzaamheden.

De door het bijzondere soort produktiesysteem, nl. een informatiesysteem te produceren informatie en het daaruit volgende informatieprodukt dienen uiteraard tegemoet te komen aan informatiebehoeften, zoals deze in de organisatie bestaan. We zullen dit, hoe vanzelfsprekend ook, toelichten aan een voorbeeld.

Een afdelingschef ontdekt dat niet tijdig afleveren van orders vrij vaak voorkomt. Om hierin verbetering te brengen zal hij moeten beschikken over bepaalde informatie. Deze informatiebehoefte (kortweg aangegeven met I_1) zou bijv. het volgende kunnen omvatten: het aantal nodige manuren per order per vakgroep en het aantal beschikbare manuren per order per vakgroep. Als echter de afdelingschef als te leveren informatie opgeeft: aantal nodige manuren per order en aantal beschikbare manuren per order, dan voldoet de gevraagde informatie, verder aangegeven met P_1 , niet aan de werkelijke informatiebehoefte I_1 . Het is immers mogelijk dat in totaal het aantal beschikbare manuren per order gelijk is aan het aantal nodige manuren per order, maar dat in sommige vakgroepen een overschot en in andere vakgroepen een tekort aan manuren is. Het probleem moge zo eenvoudig lijken dat nauwelijks voor te stellen is dat het kan optreden. Men bedenke echter dat het hier kort en gestileerd wordt weergegeven en daardoor de eenvoud onmiddellijk in het oog springt. De praktijk wijst echter uit dat door de ingewikkelde context, waarin dergelijke zaken optreden, eenvoudige zaken dikwijls moeilijk als zodanig zijn te onderkennen.

Een variant op het bovenstaande probleem is het volgende. De

informatiebehoefte I_2 is zoals boven beschreven plus de vraag voor welke combinaties van order en vakgroep geldt dat het verschil tussen benodigde en beschikbare manuren meer dan 10% van het aantal benodigde manuren bedraagt.

Geleverd wordt de informatie bestaande uit de opgave van benodigde en beschikbare manuren per combinatie order en vakgroep. Dit informatieprodukt van P_2 is het juiste produkt voor informatiebehoefte I_1 , maar niet voor I_2 . Immers men zal nu nog moeten bepalen voor welke van de combinaties geldt dat het aantal beschikbare manuren minder is dan 90% van het aantal benodigde manuren. Geen nood, zou iemand kunnen opmerken; want uit de geleverde informatie is dit laatste wel te destilleren. Immers men kan dit voor alle combinaties uitrekenen, gegeven P_2 . Inderdaad is hier een kenmerkend verschil tussen P_1 (bedoeld voor I_1) en P_2 (bedoeld voor I_2). Waar P_1 niet geschikt was te maken voor I_1 , kan P_2 door een nabewerking wel geschikt worden gemaakt voor I_2 . In deze zin is P_2 een beter produkt te noemen dan P_1 . Maar daarmee is P_2 nog geen goed produkt. We gaan hierbij nog maar voorbij aan de vraag of dergelijke nabewerkingen altijd mogelijk zijn.

Hier treedt duidelijk het verschil tussen gegevens en informatie naar voren, of in de produktie-beeldspraak tussen grondstoffen en halffabrikaten enerzijds en eindprodukten anderzijds. P_2 is in feite een halffabrikaat voor I_2 (en eindprodukt voor I_1). Waar P_1 een onbruikbaar informatieprodukt is voor I_1 , kan P_2 een onafgewerkt informatieprodukt voor I_2 worden genoemd.

Na het bovenstaande wordt het aan de lezer overgelaten een voorbeeld te bedenken, waarbij het informatieprodukt meer is dan door de informatiebehoefte wordt geëist.

Uitgangspunt voor het opstellen van een informatiesysteem moet dus zijn de informatiebehoefte. Het is dan ook belangrijk dat elke informatiebehoefte volledig en duidelijk wordt geformuleerd en daardoor verstaanbaar wordt voor anderen (en voor de informatiebehoevende zelf!). De problematiek van het duidelijk formuleren is niet eenvoudig en wordt dikwijls nog extra verzwaard door de moeilijke communicatie tussen informatie-klant (gebruiker) en informatie-producent (informatica-functionaris). Dit heeft geleid tot de creatie van een nieuwe functie, nl. van informatie-analist (wat verwarrend ook wel systeem-analist genoemd), waarvan verwacht wordt dat hij de communicatiekloof tussen beide eerder genoemden kan overbruggen.

Informatiebehoeften kunnen sterk uiteenlopen. Tussen 'gelijksoortige' organisaties zijn dikwijls al grote verschillen, enerzijds omdat de gelijksoortigheid maar tot bepaalde hoogte opgaat (denk bijv. aan de talloze variaties in organisatie-opzet), anderzijds omdat de procedures zelfs bij gelijke organisatievorm nog sterk uiteen kunnen lopen (verschillen in orderbehandeling, administratie, personeelsbeheer,

enz. en daarmee samenhangende rapportage). Ook tijd en plaats kunnen een belangrijke rol spelen bij het bepalen van de (veranderende) informatiebehoefte. Zo zal bijv. een in de tijd evoluerende sociale wetgeving de informatiebehoefte drastisch kunnen wijzigen.

Het is dan ook niet verwonderlijk dat 'standaardpakketten' (programma's voor bijv. financieel beheer, voorraadbeheer, enz.) dikwijls veel aanpassing behoeven, zo ze al niet totaal onbruikbaar zijn.

Tot slot van deze beschouwingen nog het volgende. Zoals reeds gezegd is de informatiebehoefte bepalend voor een informatiesysteem. Tussen de eerste signalering van informatiebehoefte en de uiteindelijke vervulling van deze behoefte met een informatieprodukt zullen dikwijls in de opzet van het informatiesysteem vele schakels moeten worden betrokken. Als schakels kunnen worden genoemd:

- diverse bedrijfsafdelingen (informatiebehoevenden)
 - tussenpersonen (bijv. informatie-analisten)
 - systeemontwerpers
 - programmeurs
 - operateurs
- (wanneer een computer nodig is)

Zowel binnen elke schakel als (vooral bij communicatie) tussen de schakels zal de specificatie van de gewenste informatie onverminkt moeten blijven. Dit is in de praktijk geen eenvoudige zaak. Goede schriftelijke documentatie is daarbij onontbeerlijk!

Uit het voorgaande moge ook duidelijk zijn dat er verschillende soorten informatiesystemen zijn. In de tijd gezien is een zekere evolutie te constateren, samenhangend met het doordringen van rekenautomaten in organisaties. Tot ongeveer 1965 werden rekenautomaten eigenlijk alleen gebruikt voor de 'automatisering' van het oude handwerk, zoals dat in organisatorisch gescheiden delen van een bedrijf of instelling volgens in vele decennia ontwikkelde methoden gegroeid was. De veranderingen waren eigenlijk alleen dat het menselijk schrijfvermogen werd vervangen door een vele malen snellere regeldrukker en dat menselijke reken- en sorteervermogens eveneens door een rekenautomaat werden overgenomen. Tijdwinst en een sterk gereduceerde kans op fouten waren veelal de enige pluspunten vergeleken met het oude handwerk, arbeidsbesparing werd maar zelden bereikt doch wel trad een verschuiving van arbeidsplaatsen in de richting van een rekencentrum op. Iedere toepassing was uniek in de zin dat het zijn eigen groepering van gegevens gebruikte.

Na 1965 is een versnelde ontwikkeling in 'geïntegreerde' informatiesystemen waar te nemen. Duidelijk aanwijsbare oorzaken hiervoor zijn:

- a. gebruik van betere programmeerconcepten en betere programmeertalen;
- b. de spectaculaire ontwikkeling van rekenautomaatsystemen, onder meer tot uitdrukking komend in
 - grotere en sneller toegankelijke geheugens,

- lagere kosten per eenheid van gegevensopslag en per elementaire bewerking,
 - koppeling van rekenautomaten met uiteenlopende typen input/output apparatuur via o.a. telefoonlijnen;
- c. een snel toegenomen inzicht in de gebruiksmogelijkheden van rekenautomaten.

Deze ontwikkeling geeft dus de mogelijkheid tot geavanceerde informatiesystemen, waarmee aan meer complexe informatiebehoeften kan worden voldaan. In hoeverre van deze mogelijkheid gebruik kan worden gemaakt, hangt af van de kwaliteit van

- de informatie-analyse, die moet leiden tot de precieze bepaling van de informatiebehoefte en de daarvoor benodigde gegevens,
- de gegevensstructuren en de verwerkingsprogramma's, waarmee uiteindelijk aan de informatiebehoefte moet worden voldaan.

Dit boek handelt over het laatste, dus over structurering en verwerking van gegevens. Hierbij zij opgemerkt dat structurering en verwerking van gegevens niet los van elkaar gezien kunnen worden. Immers gegevensstructuren hebben alleen maar zin als zij de kwaliteit van de gegevensverwerking bevorderen. Aan de opzet van verwerkingsprogramma's zal dan ook telkens de nodige aandacht worden besteed, mede ter verhoging en toetsing van het inzicht in gegevensstructuren.

In dit boek wordt ervan uitgegaan dat de informatie-analyse reeds heeft plaatsgevonden. Hiermee is zeker niet gezegd, dat informatie-analyse een eenvoudige zaak is. Integendeel. Het is niet voor niets dat gedurende de laatste jaren vele methoden en technieken op dit gebied worden ontwikkeld. Voor een uitgebreid overzicht hiervan zij verwezen naar een in de literatuurlijst opgenomen rapport (WDBC).

Dit eerste hoofdstuk wordt afgesloten met een paragraaf over achtergrondgeheugens. Deze zijn nl. het opslagmedium voor (grote) 'permanente' gegevensverzamelingen. Deze paragraaf zal deels een herhaling zijn van reeds bekende stof. In ieder geval zal voor volgende hoofdstukken er vanuit worden gegaan dat de begrippen van paragraaf 1.3 met appendix I geen moeilijkheden meer opleveren.

1.3 ACHTERGRONDGEHEUGENS (BACKING STORE)

Terwijl oorspronkelijk in hoofdzaak magneetbanden en in mindere mate magneettrommels als achtergrondgeheugen gebruikt werden, kwamen daar sinds ongeveer 1965 ook bij schijfgeheugens, magneetstripgeheugens, 'langzame kerngeheugens' en over enkele jaren wellicht optische en nieuwe (elektro-)magnetische geheugenssystemen. Deze ontwikkeling zal nog wel vele jaren doorgaan, aangezien men nog steeds niet tevreden is met de huidige vormen ten aanzien van o. a.

- capaciteit (hoeveel gegevens kan men in een achtergrondgeheugen onderbrengen?)
- toegankelijkheid (hoe lang duurt het voordat een hoeveelheid gegevens voor bewerking naar/van het werkgeheugen overgebracht is; kan men een bepaald gegeven direct opvragen of moet men eerst alle daarvoor liggende gegevens inspecteren?)
- betrouwbaarheid (t. a. v. storingen, sabotage, reproduceerbaarheid)
- kosten
 - van opslag (daarbij rekening houdend met de kosten van het eigenlijke opslagmedium en van de apparatuur waarmee dit medium aan de rekenautomaat gekoppeld is)
 - van toegang tot de vastgelegde informatie.

In de appendix worden eigenschappen van achtergrondgeheugens besproken. Hier zullen we alleen ingaan op enkele facetten van het gebruik van achtergrondgeheugens, die voor alle typen geheugens relevant zijn.

Een eerste facet is dat de toegangstijd tot een bepaald gegeven op een achtergrondgeheugen opgebouwd is uit (tenminste) twee delen. In de eerste plaats een periode van (afhankelijk van de soort apparatuur) ongeveer 1 tot 500 ms waarbinnen de apparatuur zich 'instelt' om gegevenstransport te gaan verzorgen en vervolgens een periode van ongeveer 1 ms (afhankelijk van de snelheid van het werkgeheugen) waarin het eigenlijke gegevenstransport plaatsvindt. Aangezien deze tijden een factor duizend groter zijn dan de verwerkingstijd van een gegeven in het werkgeheugen en men voor een bepaald karwei als regel veel meer dan één gegeven nodig heeft, zijn achtergrondgeheugens zo geconstrueerd dat met één machine-opdracht niet één gegeven maar een z.g. blok van gegevens tussen werkgeheugen en achtergrondgeheugen getransporteerd wordt (volledigheidshalve zij echter vermeld dat voor enkele zeer snelle typen trommelgeheugen niet een blok maar een enkel gegeven overgebracht wordt). Op deze manier wordt de 'insteltijd' (waarin naast mechanische bewegingen als starten en stoppen ook gecontroleerd wordt of een gegevenskanaal tussen achtergrond- en werkgeheugen niet al door een ander gegevenstransport in beslag is genomen en of het transport wel goed verlopen is) als het ware uitgesmeerd over het transport van vele in plaats van één gegeven.

Als op bovenstaande wijze de gemiddelde toegangstijd per gegeven zoveel mogelijk is gereduceerd, dan hebben we daarmee nog niet gegarandeerd dat de centrale processor optimaal wordt benut. Hiermee raken we aan:

Een tweede facet, nl. dat van het op elkaar afstemmen van I/O-tijd en processortijd. Zoals boven opgemerkt vergt een gegevenstransport van en naar de huidige achtergrondgeheugens tenminste enkele tientallen ms (I/O-tijd). De eigenlijke rekenbewerkingen en de daar-

mee gepaard gaande gegevenstransporten naar en van het rekenkundig orgaan spelen zich tegenwoordig af in microsec (processortijd). Er zal dan ook alleen sprake kunnen zijn van een evenwicht tussen I/O- en processortijd, als op de gegevens in het werkgeheugen een groot aantal (duizenden) bewerkingen moet plaatsvinden; men spreekt dan van een 'balanced run'. Is de benodigde I/O-tijd groter dan de processortijd, dan spreekt men van een 'I/O limited run'; in het omgekeerde geval van een 'processor limited run'.

Bij bewerkingen op bestanden zullen we vrijwel altijd te maken hebben met 'I/O limited runs'; de processor zou het grootste deel van de tijd niet actief zijn. Om dit te ondervangen heeft men in de loop der tijd vele technieken geïntroduceerd (met als doel een 'balanced run' te krijgen).

Wij zullen hier de techniek van het zogeheten autonome transport bespreken. Deze techniek veronderstelt dat de werkgeheugens zo zijn geconstrueerd, dat terwijl een deel ervan 'samenspeelt' met de processor, een ander deel vrijwel tegelijkertijd kan communiceren met het achtergrondgeheugen en de invoer/uitvoerorganen. Met de techniek van autonoom transport is het dus mogelijk om verschillendsoortige bewerkingen simultaan te doen plaatsvinden. Om deze gelijktijdigheid tot stand te kunnen brengen, maakt men gebruik van buffers, d. w. z. aparte geheugentrajecten ter lengte van de gegevenshoeveelheid die betrokken is bij het gegevenstransport van/naar het achtergrondgeheugen. Heeft men per randapparaat (tenminste) twee buffers ter beschikking, dan heeft men dankzij de autonomie van transport de mogelijkheid om één buffer te vullen vanuit het achtergrondgeheugen en met de inhoud van de andere buffer te rekenen. Is men met het laatste klaar, dan verwisselt men de rol van de twee buffers, enz. De vrome hoop is hierbij natuurlijk dat het rekenen met de inhoud van een buffer net iets langer duurt dan het vullen van een buffer, zodat men schijnbaar niet anders doet dan rekenen. Rekent men heel veel langer dan de tijd nodig voor het vullen van een buffer, dan is het eigenlijk niet nodig om een tweede buffer te gebruiken omdat gegevenstransport dan maar een kleine toeslag op de rekentijd betekent. Is de rekentijd daarentegen heel veel korter dan de gegevenstransporttijd, dan haalt het rekenproces ook bij meer dan twee buffers onherroepelijk het vullen van de buffers in, waarna de buffervultijd bepalend is voor de snelheid van het proces. Uit het laatste volgt ook dat het gebruik van meer dan twee buffers alleen dan zinvol is, wanneer er een vrij sterke fluctuatie is in de gemiddelde rekentijd per bufferinhoud.

Door bepaalde beslissingen kan men de rekentijd op nuttige wijze 'vergroten', wanneer deze te klein is t. o. v. de buffervultijd. Zonder op details in te gaan noemen we:

- met de in een buffer aanwezige gegevens meer berekeningen uitvoeren door meer taken in één programma te verenigen. Nadelen hiervan zijn echter: ingewikkelder probleemanalyse, moeilijker te

- onderhouden programma's, meer geheugenruimte voor het programma;
- het gebruiken van ingewikkelder, maar minder ruimte vergende gegevensstructuren; immers het isoleren van gegevens uit deze structuren kost meer tijd, terwijl de gegevenstransporttijd afneemt door grotere compactheid;
 - fysieke vastlegging van gegevens in een vorm die optimaal is voor invoer/uitvoer, doch conversietijd vergt als er mee gerekend moet worden (bijv. binair gecodeerde decimale representatie in plaats van binaire representatie van getallen);
 - de gegevens zo te structureren dat alleen het 'actief gebruikte' deel ingelezen en verwerkt wordt. Nadelen hiervan zijn weer ingewikkelder probleemanalyse en dat 'hoge activiteit' vaak een tijdelijk iets is.

Een derde facet is dat ter verlaging van apparatuurkosten de verschillende achtergrondgeheugens niet individueel gekoppeld zijn aan het werkgeheugen. Meestal is een aantal gelijksoortige geheugens gekoppeld aan een z.g. 'besturingseenheid' ('control unit'), die een aantal functies voor de aangesloten geheugens behartigt. Een aantal besturingseenheden is op zijn beurt weer gekoppeld aan een tamelijk kostbaar gegevenskanaal naar het werkgeheugen. Door dit gegevenskanaal vindt het eigenlijke gegevenstransport plaats, terwijl ook besturingssignalen naar de besturingseenheden en eventueel ook de achtergrondgeheugens door het gegevenskanaal getransporteerd kunnen worden. Op de details van de werking van de verschillende eenheden, die met elkaar de 'configuratie' van de apparatuur bepalen, zullen we echter verder niet ingaan.

Wel zij erop gewezen dat de aanwezigheid van kanalen ook de zogeheten multiprogrammering mogelijk maakt. Immers, terwijl met gegevens van een programma in een deel van het werkgeheugen gerekend wordt, worden voor een ander programma delen van het werkgeheugen via de kanalen gevuld/weggeschreven uit/naar het achtergrondgeheugen. Zijn de kanalen echter druk bezet dan kunnen 'wachttijdproblemen' optreden. De toegangstijden voor gegevens, die volgen uit de elektromechanische constructie van achtergrondgeheugens, kunnen dan wel eens helemaal niet maatgevend zijn voor de tijd die nodig is om de gewenste gegevens naar het werkgeheugen te krijgen.

Modelmatig kunnen we dit vergelijken met de situatie die ontstaat voor een loket waar zich mensen melden om aan het loket een bepaalde afhandeling te krijgen. Is de tijd die gemiddeld verloopt tussen het binnenkomen van twee mensen (de aanbiddingstijd) zeer groot vergeleken met de (gemiddelde) afhandelingstijd, dan kan men verwachten dat de wachttijd zeer klein is en de totale verblijftijd voor het loket vrijwel gelijk is aan de afhandelingstijd. Naarmate echter de aanbiddingstijd relatief t. o. v. de afhandelingstijd afneemt, zal

de gemiddelde wachttijd toenemen. Dat deze toename drastische vormen kan aannemen, blijkt uit de wachttijdtheorie, waar wordt afgeleid dat onder bepaalde voorwaarden (negatief exponentiële verdeling van aanbiedings- en afhandelingstijd) de verhouding van gemiddelde wachttijd en afhandelingstijd gegeven wordt door

$$E_w / E_a = g / (1-g)$$

waarbij E_w : gemiddelde wachttijd

E_a : gemiddelde afhandelingstijd

g : bezettingsgraad = gem. afhandelingstijd / gem. aanbiedingstijd

We zien dat voor waarden van g tot $2/3$ de (gemiddelde) wachttijd hoogstens het dubbele wordt van de gemiddelde afhandelingstijd, doch voor $g > 2/3$ schrikbarend toeneemt (vergelijk de situatie bij een iets groter verkeersaanbod op toch al drukke wegen).

Het is dus zaak om bij het ontwerpen van informatiesystemen er zoveel mogelijk voor te zorgen dat de aanbiedingstijd groot blijft t.o.v. de afhandelingstijd. Dit wordt meestal bereikt door de per keer aan te vragen informatiehoeveelheid niet te klein te maken.

OPGAVEN

- 1.1. De directie van een school voor HBO wil aan het eind van een cursusjaar graag zo concreet mogelijke informatie over eerstejaars. Zij wil met name:
 - graag weten hoe groot het percentage eerstejaars is dat bevorderd wordt naar het 2e jaar;
 - dit percentage ook nog splitsen naar eerstejaars met HAVO-vooropleiding en andere vooropleidingen;
 - vervolgens de HAVO-eerstejaars nog splitsen naar HAVO met wiskunde en HAVO zonder wiskunde;
 - tenslotte deze cijfers ook nog graag zien gesplitst naar toeleverende school (er zijn 10 van deze toeleverende scholen).
 - a. Ligt met het bovenstaande de informatiebehoefte duidelijk en ondubbelzinnig vast? Zo nee, geef dan een goede beschrijving van de informatiebehoefte. Deze beschrijving mag niet in strijd zijn met de reeds duidelijke punten uit bovenstaande beschrijving.
 - b. Maak een schets van het informatieprodukt dat aan de directie moet worden geleverd.
- 1.2. Maak een schets van de informatie, zoals gevraagd volgens I₂ (par. 1.2).

- 1.3. Bedenk een (eenvoudig) voorbeeld, waarbij het informatieproduct meer is dan door de informatiebehoefte wordt geëist.
- 1.4. Waarom is het gebruik van meer dan twee buffers alleen zinvol, wanneer er een sterke fluctuatie is in de rekentijd per bufferinhoud?
- 1.5. Gegeven is een bestand op magneetband van 80000 records van elk 200 karakters. Gevraagd wordt als functie van de blokkingsfactor:
- a. de lengte van het bestand (in meters)
 - b. de totale leestijd (in sec)
- Zet deze functies uit in grafieken.
Voor de band geldt:
IBG = 1,5 cm
dichtheid : 500 kar/cm
bandleessnelheid : 50 kc/sec
start- + stoptijd per IBG : 10 millisecon
totale bandlengte : 700 m.
- 1.6. Een studentenbestand bestaat uit 10000 records. Elk record heeft een vaste lengte van 720 karakterposities. Dit bestand is opgeslagen op magneetband. Voor deze magneetband geldt:
dichtheid : 800 kpi (karakters per inch)
lees/schrijfsnelheid : 75"(inch)/sec
IBG : 0.6"
overbruggen IBG (incl. start- en stoptijd) : 10 millisecon
lengte magneetband : 2500 ft (1 ft = 12 inch)
terugspoeltijd (rewind time) voor gehele band : 80 sec.
Verder is de maximale grootte van de invoerbuffer in het werkgeheugen 750 posities.
- a. Hoe groot is de snelheid bij terugspoelen? (in inches/sec)
 - b. Hoeveel banden zijn voor dit bestand nodig?
 - c. Hoeveel tijd is nodig voor het lezen (inclusief terugspoelen) van het hele bestand?
 - d. Hoeveel tijd zou nodig zijn om het hele bestand te lezen (inclusief terugspoelen) als het aaneengesloten op de band zou staan?
 - e. Als het bestand op schijf zou worden opgeslagen (met 200 cylinders per disc-pack; 10 sporen per cylinder) en elk spoor 4 records à 720 tekens kan bevatten, hoeveel packs zouden dan nodig zijn voor dit bestand?
- 1.7. a. In de beschrijving van een disc storage system komt het volgende voor:
'Each disc storage unit (DSU) may contain up to 16 discs,

providing 32 recording surfaces. Each surface has two zones with 128 tracks each. Inner tracks can contain 320 words each, outer tracks, 640 words each. Thus, each DSU has a capacity of up to 23, 592, 000 6-bit characters.'

Kunt u instemmen met de laatste zin?

- b. In een andere beschrijving lezen we:
- 'Since the read/write mechanism is of fixed head-per-track design, access time to all disc locations is constant.'
- Wat wordt bedoeld met deze zin?

2 BASISBEGRIPPEN BIJ GEGEVENSSTRUKTUREN

2.1 OBJEKTEN EN ATTRIBUTEN; LOGISCHE STRUKTUUR

In een informatiesysteem moeten meestal grote aantallen gegevens worden verwerkt, variërend van enkele duizenden tot enkele miljoenen. Deze grote aantallen vragen om een zorgvuldige en systematische structurering en verwerking. De hierbij gebruikte terminologie zal nu eerst behandeld worden.

De informatiebehoefte van een organisatie heeft betrekking op de voor deze organisatie van belang zijnde 'objecten'*). Voorbeelden hiervan worden in onderstaand tabelletje gegeven, waarbij vanzelfsprekend niet getracht is bij elke organisatie een uitputtende opsomming van relevante objecten te geven.

VOORBEELD 2.1.1

<u>Organisatie</u>	<u>Relevante objecten</u>
Onderwijsinstelling	Studenten Docenten Vakken Afdelingen
Ziekenhuis	Patiënten Afdelingen Specialisten Medicijnen
Handelmaatschappij	Leveranciers Klanten Artikelen Vertegenwoordigers

*) In plaats van het woord 'object' wordt (en werd ook in vorige drukken van dit boekje) het woord 'entiteit' gebruikt. Wij geven (nu) de voorkeur aan het woord 'object', omdat dit naar onze mening suggestiever is en tevens meer in overeenstemming is met de hedendaagse literatuur.

Produktiebedrijf

Afdelingen
 Werknemers
 Machines
 Produkten
 Leveranciers
 Afnemers
 Orders

Gemeente

Bewoners
 Ambtenaren
 Straten

Voor ieder objekt zijn talloos vele 'kenmerken' op te noemen; bijvoorbeeld voor het objekt student: registratienummer, naam, adres, woonplaats, geboortedatum, soort vooropleiding, jaar eindexamen vooropleiding, cijferlijst vooropleiding, lengte, schoenmaat, vakantieadres, kleur ogen, gewicht, studie-afdeling, naam vader, beroep vader, naam moeder, beroep moeder, aantal vrienden, aantal vriendinnen, aantal ooms, aantal tantes. Deze lijst van kenmerken is nog gemakkelijk uit te breiden. De relevantie van (het registreren van) een kenmerk is weer afhankelijk van de informatiebehoefte van een organisatie. Zo zal een opleidingsinstituut voor een student kenmerken als naam, adres, woonplaats, geboortedatum, soort vooropleiding en nog enkele van bovengenoemde kenmerken wel relevant vinden, maar niet geïnteresseerd zijn in bijv. lengte, schoenmaat, aantal vrienden e.d. Het bepalen van relevante kenmerken vormt een onderdeel van de informatie-analyse, misschien wel het moeilijkste onderdeel van deze analyse. Zoals reeds in paragraaf 1.2 werd opgemerkt valt de informatie-analyse buiten het bestek van dit boek. Wij zullen dus in voorkomende gevallen (voorbeelden en opgaven) veronderstellen dat bekend is welke kenmerken relevant zijn voor een objekt.

In plaats van het woord 'kenmerk' wordt tegenwoordig in het kader van gegevensstructuren meestal het woord 'attribuut' gebruikt. Wij zullen ons bij dit veelvuldige gebruik aansluiten*).

Om over een objekt en zijn attributen te kunnen praten zullen we een 'naam' gebruiken voor het objekt en voor elk van de attributen, zoals wij trouwens in voorbeelden in het voorgaande reeds deden. Bijv.: student (registratienummer, naam, adres, woonplaats, geboortedatum, soort vooropleiding).

Uiteraard is men voor een bepaald objekt of attribuut vrij in het kiezen van een naam, maar men moet uiteraard wel consequent

*) In vorige drukken van dit boek werd het woord 'item' gebruikt in plaats van 'kenmerk'. Alternatieve benamingen, die men in de literatuur kan tegenkomen zijn: data-item, elementary item, data-element, property.

dezelfde naam blijven gebruiken voor dat object of attribuut. Zo is het wat korter om bovenstaande beschrijving van student als volgt weer te geven:

(2) student (rnr, nm, adr, wpl, gbd, sv).

Hierbij zal het dan vooral ten behoeve van anderen wel nodig zijn een lijst bij te houden ter verklaring van de gebezigde afkortingen.

Een groep van attributen kan men voorzien van een groepsnaam. Een bekend voorbeeld (uit de administratieve sector) is de combinatie van (naam, adres, woonplaats), waarvoor de groepsnaam naw gebruikelijk is. Een ander voorbeeld is de groepsnaam fk (fysieke kenmerken) voor de combinatie van (lengte, gewicht, kleur ogen).

In de beschrijving van een object worden groepsnamen opgenomen zoals in onderstaand voorbeeld:

(3) student (rnr, naw(nm, adr, wpl), gbd, sv).

Of een groep en dus de elementen van een groep als zodanig herkenbaar moeten zijn, is afhankelijk van de informatiebehoefte van de onderhavige organisatie. Als geen van de elementen van een groep ooit als zodanig beschikbaar hoeft te zijn, heeft het geen zin die elementen afzonderlijk te benoemen. Als bijv. naam, adres, woonplaats alleen maar in deze combinatie van drie worden gebruikt, dan is beschrijving (4) in plaats van (3) voldoende:

(4) student (rnr, naw, gbd, sv).

Omgekeerd kan het nodig zijn een attribuut als een groep te gaan beschouwen, als namelijk dit attribuut logischerwijze opsplitsbaar is en deze opsplitsing binnen het onderhavige informatiesysteem ook zin heeft. Zo zou een opsplitsing van gbd in (jaar, maand, dag) soms best zinvol kunnen zijn. In zo'n geval zou (4) vervangen moeten worden door (5):

(5) student (rnr, naw, gbd(jaar, maand, dag), sv).

Een heel andere wijze van groepering vindt plaats in de zogeheten repeating group. Een repeating group bestaat uit de herhaling van een bepaald 'patroon'.

VOORBEELD 2.1.2 (RG = repeating group)

- RG examenvakken, bestaande uit de (maximaal 7) namen van het eindexamenpakket. Het herhalingspatroon bestaat hier dus uit een vaknaam.
- RG examenresultaten, bestaande uit de combinatie van (vakcode, vaknaam, c1, c2, ce), waarbij
 - c1 : cijfer eerste corrector
 - c2 : cijfer tweede corrector
 - ce : eindcijfer.

Het herhalingspatroon bestaat hier dus uit de combinatie (vakcode, vaknaam, c1, c2, ce). Gebruikers van het informatiesysteem zullen

moeten uitmaken of men in deze combinatie groepen moet onderscheiden en zo ja welke groepen, zoals bijv. (vakcode, vaknaam, cijfers(c1, c2, ce)), maar ook (eindresultaat(vakcode, vaknaam, ce), c1, c2).

- RG vorige adressen, bestaande uit een niet vastliggend aantal herhalingen van het patroon: (adres, woonplaats).

Wanneer het (maximum) aantal herhalingen van een repeating group vastligt (zoals in de eerste twee voorbeelden) spreekt men ook wel van een vektor, tabel of een array.

Onze afspraken voor de beschrijving mogen duidelijk zijn uit onderstaand voorbeeld:

- (6) student (rnr, naw(nm, adr, wpl), RG vor-adr(adr, wpl), gbd, sv, RG(7) ex-res(vkc, vknm, c1, c2, ce)).

Een repeating group wordt dus beschreven met

- RG, gevolgd door het
- aantal herhalingen als dit aantal vastligt, en de naam gevolgd door het
- herhalingspatroon tussen ronde haakjes geplaatst.

In dit hoofdstuk hebben wij ons tot nu toe bezig gehouden met wat genoemd wordt de logische structuur van de gegevens, m.a.w. met de opsomming van relevante objecten en attributen, zonder ons voorsnog te bekommeren om zaken van opslag van gegevens en fysieke geheugenmedia. Hierop zullen wij overigens nog uitgebreid ingaan. Over de logische structuur van de gegevens wordt met het voorgaande nog maar een summier, onvolledige inleiding gegeven. Voor de volgende hoofdstukken van dit boek is dit echter voorlopig voldoende. In het laatste hoofdstuk zal nog iets nader op de logische structuur worden ingegaan. Met name zal daar de zogeheten normalisatie (van objecten) worden besproken, waarmee systematisch kan worden vastgelegd welke attributen bij welke objecten behoren. Voor de rest van het boek doet dit echter niet ter zake en zullen we uitgaan van gegeven objectbeschrijvingen in vormen als hierboven.

2.2 TYPEN EN INDIVIDUEN

In de voorgaande paragraaf hebben we van de daar behandelde objecten geen individuele 'gevallen' besproken. Zo is van het object student geen bepaalde student ter sprake gekomen. De beschrijvingen, die wij gaven, gelden dan ook niet voor een bepaalde student, maar voor elke student. Zo'n beschrijving noemen we een type-beschrijving en wij spreken in dit verband dan ook wel van een objecttype. De bijbehorende attributen worden uiteraard ook attribuuttypen genoemd.

Een objecttype dient dus goed te worden onderscheiden van een

individueel optredende 'waarde' van dit type. In navolging van de hedendaagse literatuur zullen we meestal niet over waarde spreken maar over (objekt-)occurrence*). Voor de attributen wordt echter niet het woord attribuut-occurrence maar attribuutwaarde gebruikt. Een objekt-occurrence wordt dus beschreven door de bijbehorende attribuutwaarden. Een objekttype wordt beschreven door de namen van de bijbehorende attributtypen. Zo is (2) in de vorige paragraaf een beschrijving van het objekttype student en (109751, P. Krijt, Omloop 6, Eindhoven, 580116, Athenaeum) een beschrijving van een objekt-occurrence van het type student.

Het onderscheid tussen type en occurrences kan ook worden weergegeven met de taalkundige begrippen soortnaam (voor type) en eigennaam (voor occurrence, individuele waarde). Bijv. soortnaam: adres; eigennaam: Omloop 6.

Ter illustratie nog enkele occurrences, behorende bij beschrijving (6) in de vorige paragraaf.

VOORBEELD 2.2.1

attributen	nr	naw			rep-gr vor-adr	
		nm	adr	wpl	adr	wpl
(occurrence 1)	109751	P. Krijt	Omloop 6	Eindhoven	Dalweg 5 Lelylaan 15	Eindhoven Amsterdam
(occurrence 2)	89162	H. Slim	Domweg 13	Geldrop		

attributen	gbd	sv	rep-gr(7) ex-res				
			vkc	vknm,	c1,	c2,	ce
(occurrence 1)	580116	Athen B	W1	Analyse	7	6	7
			W5	Algebra	9	9	9
			M	Mechanica	6	5	5
			E4	Elektronenoptica	7	7	7
			E9	Hoogspanning	5	7	6
			I2	Programmeren	9	9	9
			I7	Bestandsorganisatie	5	6	6
(occurrence 2)	600510	Athen B	W2	Optimalisering	8	8	8
			W5	Algebra	7	7	7
			W9	Maattheorie	5	6	6
			I2	Programmeren	9	8	9
			I4	Vertalerbouw	7	5	7
			I7	Bestandsorganisatie	8	7	8
			B2	Bedrijfseconomie	7	8	8

*) In vorige drukken van dit boek werd hiervoor het woord 'entry' gebruikt.

Een verzameling attributen, waarvan de waarde iedere occurrence van een objekttype uniek identificeert, noemen we een sleutel (key). Dikwijls bestaat zo'n sleutelverzameling uit slechts één attribuut, het sleutel-attribuut. Het is verder mogelijk dat meer dan één verzameling als sleutelverzameling kan fungeren. Zo kan registratienummer een sleutel zijn voor het objekttype student, maar mogelijk ook de combinatie (naam, adres, woonplaats).

Tot slot van deze paragraaf willen wij wijzen op een uitspraak van E. F. Codd, een van de grondleggers van de moderne opvattingen over gegevensstructuren. Deze waarschuwt ervoor dat vele misverstanden in gegevensverwerking terug te voeren zijn op een gebrekkig onderscheid tussen type en occurrence.

2.3 OPSLAGSTRUCTUREN

Gegevens kunnen alleen maar worden 'opgeroepen', als ze ergens liggen opgeslagen op een of ander opslagmedium. Zo'n opslagmedium zal wegens de vereiste capaciteit vrijwel altijd een achtergrondgeheugen zijn, tegenwoordig meestal in de vorm van een of meer magneetbanden of een of meer magneetschijven. Deze geheugenmedia hebben in het vorige hoofdstuk reeds genoemde eigenschappen, waarmee terdege rekening moet worden gehouden, als men de opslag tot op fysisch niveau precies wil blijven volgen. Om echter een eerste globale indruk van de opslag van occurrences te krijgen, gaat men dikwijls in eerste instantie uit van een oneindig groot achtergrondgeheugen, dat verdeeld is in geheugentrajecten van gewenste grootte. Om spoor aantallen, spoorcapaciteit, blocking, enz. bekommeren we ons niet als we spreken over opslagstructuren.

De opslag van een occurrence noemen we een record*). De verzameling van alle records van één objekttype noemen we een bestand (file).

Opslagstructuren zullen we nu op twee niveaus behandelen:

- Record-structuur, d.w.z. de opslagstructuur voor een record
- Bestand-structuur, d.w.z. de opslagstructuur voor een bestand (men gebruikt in dit verband ook vaak het woord bestandsorganisatie).

De opslagstructuur voor een record zullen wij in de volgende subparagraaf (kort) behandelen, terwijl het grootste deel van het boek verder zal gaan over de opslagstructuur voor een bestand, dus over bestandsorganisatie.

*) Soms spreekt men van een logical record in tegenstelling tot geblokte opslag in zogeheten physical records (vgl. appendix I).

2.3.1 Opslagstructuur voor een record

Een record is een weergave van een occurrence (van een objekttype). Dit betekent een weergave van de attribuutwaarden van deze occurrence. De weergave van een attribuutwaarde binnen een record noemen we een veld.

We zullen geheugenruimte dikwijls uitdrukken in bytes. Een byte is die geheugenruimte, die nodig is om één karakter (als verzamelnaam voor letter, cijfer, leesteken, symbool, enz.) op te bergen*).

De lengte van een veld is gelijk aan het aantal bytes dat voor dit veld in het geheugen gereserveerd is. De attribuutwaarde, die opgeslagen moet worden, is niet de enig bepalende factor voor de veldlengte. Uiteraard zal de lengte minstens het aantal bytes moeten omvatten dat nodig is om de attribuutwaarde weer te geven. Dit betekent bij voorbeeld 2.2.1 dat inzake het attribuuttype wpl voor de eerste occurrence een veldlengte van minimaal 9 posities nodig is en voor de tweede occurrence een veldlengte van minimaal 7 posities. Men kan nu kiezen voor een vaste of variabele veldlengte. Bij een vaste veldlengte is het veld even groot (en wel groot genoeg voor de 'langste' attribuutwaarde) voor alle records van hetzelfde objekttype. Evenzo kan men spreken van vaste of variabele recordlengte, al naar gelang alle records behorend bij eenzelfde objekttype wel of niet dezelfde lengte hebben. Er zij op gewezen dat variabele veldlengte mogelijk is bij vaste recordlengte. Evenzo verdraagt variabele recordlengte zich met vaste veldlengte (bij objekttypen met repeating groups).

Behalve de lengte van een veld is ook nog van belang de aard van het veld. Hieronder wordt verstaan de soort waarde die in het veld kan worden opgeslagen: numeriek, alfanumeriek of alfabetisch. Wij zullen hieraan in dit boek verder geen aandacht besteden, maar impliciet aannemen dat de velden geschikt zijn voor de soort waarden die er in moeten worden opgeslagen. Men zal voor ieder objekttype de lengte van de records en velden en de aard der velden moeten vastleggen (vgl. bijvoorbeeld de PICTURE-clausule in de DATA DIVISION van een COBOL-programma). Deze vastlegging heeft dan betrekking op alle records van een zeker objekttype. In dit verband spreekt men dan ook van (de specificatie van) een recordtype en bijbehorende veldtypen.

Uit het voorgaande moge duidelijk zijn dat bij een objekttype verschillende recordtypen mogelijk zijn. Als grondstructuren voor

*) Wij laten hierbij de mogelijkheid buiten beschouwing dat de benodigde geheugenruimte per karakter verschillend kan zijn bij de z. g. compacte opslag.

recordtypen zijn de volgende vier te onderscheiden (zie ter illustratie voorbeeld 2.3.1).

I. DE VASTE STRUKTUUR

1. Er is een vast aantal velden in alle records behorende bij een bepaald objekttype.
2. De velden liggen in een vaste volgorde.
3. Alle velden, behorende bij één attribuuttype, hebben dezelfde lengte.
4. Uit bovenstaande volgt dat ook de recordlengte vastligt en ook de begin- en eindpositie van ieder veld t.o.v. de beginpositie van het record.
5. In tegenstelling tot de andere drie structuren is het in deze vaste structuur dan ook niet nodig dat naast de velden nog hulpvelden worden opgenomen.

Deze structuur is door zijn eenvoud en overzichtelijkheid de meest gebruikte, niet zuinig in omvang maar wel de zuinigste in tijd.

II. DE GEINDICEERDE STRUKTUUR

- 1 en 2 als bij de vaste structuur.
3. De velden, behorende bij één attribuuttype, kunnen verschillende lengte hebben.
4. Daarom is ook de recordlengte variabel en moeten begin- en eindpositie van elk veld t.o.v. het recordbegin dus per record apart worden bijgehouden.
5. Voor dit laatste worden dan ook hulpvelden in het record opgenomen, die als wijzers (pointers, links, references) dienst doen om genoemde begin- en eindposities aan te geven. Deze wijzers worden eenvoudigheidshalve als regel in het begin van een record opgenomen.

Deze structuur kan geschikt zijn voor objekttypen, waarvan attribuutwaarden per occurrence sterk wisselen qua aantal karakters, zoals in het geval van familie- en straatnamen. Deze structuur is zuinig in omvang, maar niet in tijd.

III. DE SEPARATOR STRUKTUUR

Dit is eigenlijk een variant op de vorige structuur. De eerste vier punten van de geïndiceerde structuur gelden dan ook voor deze structuur.

5. Om begin- en eindposities van velden te bepalen worden nu geen wijzers gebruikt, maar zogeheten separatoren. Dit zijn hulpvelden, gelegen tussen de attribuutvelden. De waarde van deze hulpvelden wordt aangegeven met speciale karakters (zoals komma en puntkomma), die niet gebruikt worden voor het vastleggen van attribuutwaarden.

VOORBEELD 2.3.1 Opslagstructuren voor records

	(naam)	(naam echtgen.)	(adres)
I (fixed)	A .J A N S E N M .M .K L A A S E N P L E I N 7		
	F .J .B A L K S P O O R S T R 6		
II (indexed)	<div>wijzers</div> <div>41122430A .J A N S E N M .M .K L A A S E N P L E I N 7 </div> <div>0 1 2 3 4 12 24 30</div> <div>41121221F .J .B A L K S P O O R S T R 6 </div> <div>4 12 21</div>		
III (separator)	A .J A N S E N ,M .M .K L A A S E N ,P L E I N 7 ;	<div>separator</div>	einde record
	F .J .B A L K , S P O O R S T R 6 ;		
IV (label)	<div>labels</div> <div>N M ;A .J A N S E N ;A D R ;P L E I N 7 ;N M E G ;M .M .K L A A S E N ; ;</div> <div>A D R ;S P O O R S T R 6 ;N M ;F .J .B A L K ; ;</div>		einde record

Deze structuur is zeer zuinig in omvang, maar in tijd nog wat minder dan de vorige.

IV. DE LABEL STRUKTUUR

Dit is in feite de meest vrije structuur en daarmee ook de meest bewerkelijke (minst zuinig in omvang en tijd).

1. Er ligt nu ten aanzien van aantal, lengte en volgorde der velden niets vast.
2. Bij elke attribuutwaarde zal dan ook vermelding van de attribuutnaam (label) nodig zijn.
3. Uiteraard is de recordlengte variabel.

Mengvormen van deze structuren komen ook voor: vooral van de eerste twee, waarbij men door 'opvullen' (padding) toch een vaste recordlengte nastreeft.

2.3.2 Opslagstructuur voor een bestand. Algemene inleiding

Bestandsorganisatie houdt in dat moet worden vastgelegd hoe records onderling geordend zijn. De keuze van een bepaalde organisatie wordt bepaald door de aard van een toepassing, dus door de eisen van het informatiesysteem.

Om aan de informatiebehoefte op de juiste wijze te kunnen voldoen (dus het gevraagde informatieprodukt te kunnen leveren) moet men uiteraard kunnen beschikken over actuele (up-to-date) en betrouwbare gegevens. Dit is alleen maar te realiseren door een goed onderhoud van de bestanden (file maintenance). Dit onderhoud gebeurt in feite door het tijdig en juist uitvoeren van de volgende drie bewerkingen.

- Invoe-gen (inserting) van records aan een bestand. (Het opbouwen van een bestand is hiervan eigenlijk een bijzonder geval.)
- Veranderen (updating) van delen van een of meer records.
- Verwijderen (deleting) van een of meer records.

In de literatuur is men overigens niet eenstemmig over de betekenis van de juistgenoemde termen. Zo wordt onder updating wel eens verstaan de verzameling van de drie genoemde bewerkingen.

Ook het woord muteren wordt wel gebruikt als verzamelnaam voor de drie genoemde bewerkingen. In dit boek zullen wij deze betekenis voor muteren aanhouden. Voor het muteren van een bestand zijn mutatiegegevens nodig. Als sprake is van verschillende mutaties, dan zullen de mutatiegegevens worden opgeslagen in een zogeheten mutatiebestand (transaction file).

In tegenstelling tot het mutatiebestand noemt men het te onderhouden bestand een stambestand (master file). Een muta-

tiebestand heeft uiteraard een tijdelijk karakter; na verwerking van de mutaties is het eigenlijk niet meer nodig. Dikwijls wordt een mutatiebestand nog enige tijd bewaard voor eventuele reconstructie-doeleinden bij het optreden van calamiteiten. Een stambestand heeft een meer permanent karakter en wordt dan ook wel permanent bestand genoemd.

We zullen in het verdere verloop van dit boek ervaren dat genoemde onderhoudstaak niet moet worden onderschat. Een goed onderhouden bestand is dan verder geschikt voor gebruik. Dit gebruik kan nodig zijn voor verschillende doeleinden. We noemen o. a.

- raadpleging, bijv. raadpleging van tentamenresultaten in studentenrecords;
- kopiëring, bijv. een lijst van alle studentenrecords, door het studentenbestand te kopiëren via de regeldrukker;
- berekening, bijv. prijs en verkochte hoeveelheid uit een artikel-record om het brutobedrag te berekenen.

Uiteraard zullen ook dikwijls combinaties van gebruiksvormen voorkomen. Zo zal kopiëring en berekening dikwijls pas kunnen plaatsvinden na voorafgaande raadpleging.

In samenhang met bestandsverwerkingen voor onderhoud en/of gebruik worden de volgende begrippen gebruikt.

- **bestandsactiviteit** (file activity). Hiermee wordt bedoeld het percentage records dat van het totaal aantal records van een bestand wordt gebruikt bij een toepassing (programma, run). De activiteit van een bestand wordt dus niet door het bestand zelf bepaald maar door de toepassing, waarbij dit bestand wordt gebruikt. Zo kunnen, bij verschillende toepassingen, verschillende activiteitspercentages voor eenzelfde bestand gelden. Zelfs kan dit percentage voor eenzelfde herhaalde toepassing wisselen. Zo kan een dagelijkse updating van een bestand de ene dag betrekking hebben op 60% van de records, een andere dag op 72%, etc.
- **bestandsverandering** (file volatility), ook wel: veranderingsgraad genoemd. Hiermee wordt bedoeld het percentage records dat per periode een verandering in een of meer velden ondergaat. Een bestand met een hoge activiteit per verwerkingsgang kan een lage veranderingsgraad (per periode) hebben, als tijdens de verwerking vrijwel uitsluitend raadpleging plaatsvindt.
- **bestandsverloop** (file turnover), ook wel: vervangingsgraad genoemd. Hiermee wordt bedoeld het aantal records dat per periode wordt verwijderd en vervangen door nieuwe records (vergelijk: personeelsverloop). Dit aantal wordt meestal weergegeven als percentage van het totaal aantal records aan het begin van een periode. Als bijv. aan het begin van een maand een bestand 10000 records bevat en gedurende deze maand 1000 records worden verwijderd en 1000 nieuwe records worden toegevoegd, dan is het

bestandsverloop over deze maand 10%.

- bestandsgroei (file growth). Hiermee wordt bedoeld de (procentuele) verandering in het totaal aantal records per periode. In het zojuist gegeven voorbeeld is de bestandsgroei dus 0%. Bij verwijdering van 1000 records en toevoeging van 1500 records zou de bestandsgroei 5% bedragen.

Ook over bovenstaande vier bestandsbegrippen bestaat in de literatuur geen eensgezindheid betreffende hun betekenis. Zo wordt bijv. volatility ook gebruikt in de betekenis van de hier beschreven turnover.

Er zijn twee grondvormen van bestandsorganisatie:

- de sequentiële bestandsorganisatie
- de directe bestandsorganisatie

Bij sequentiële bestandsorganisatie zijn de records in (vrijwel altijd) oplopende sleutelvolgorde bereikbaar. De (relatieve) plaats van een record in een sequentieel georganiseerd bestand is dus afhankelijk van het aantal records, dat een lagere sleutelwaarde heeft dan het onderhavige record. Aangezien in het algemeen niets vastligt over dit aantal voorgaande records, is er ook geen direct verband tussen sleutelwaarde en plaats van een record in een sequentieel georganiseerd verband.

Dit ligt anders bij de directe bestandsorganisatie. Deze organisatie wordt juist gekenmerkt door het feit dat er voor elk record in het bestand een direct verband bestaat tussen sleutelwaarde (identificatie) en adres (lokatie).

In de literatuur vindt men nog vele andere vormen van bestandsorganisatie, maar deze zijn in feite terug te voeren tot een combinatie van de zojuist genoemde grondvormen. Van deze combinaties noemen we in het bijzonder:

- de index-sequentiële bestandsorganisatie
- de index-directe bestandsorganisatie.

Aan elk van deze vier organisaties zal een aparte bespreking worden gewijd.

In een sequentieel georganiseerd bestand zijn de records in het achtergrondgeheugen slechts in sleutelvolgorde bereikbaar. Deze opslag in sleutelvolgorde wordt bij deze organisatie praktisch altijd geëffectueerd door zogeheten consecutieve opslag, d.w.z. opslag, waarbij records met opeenvolgende sleutelwaarde achter elkaar gelegen zijn. Om records in een vaste volgorde te benaderen, is consecutieve opslag het meest ideaal. Het aanhouden van een vaste volgorde met behulp van (uitsluitend) consecutieve opslag kan echter de nodige complicaties bij het onderhoud opleveren. Op dit probleem zullen we verderop meermalen uitvoerig ingaan. Daarom wordt voor het vastleggen van een bepaalde volgorde van een verzameling records soms niet gekozen voor consecutieve opslag, maar voor opslag in een zo-

geheten lijststructuur. Deze structuur, die in een apart hoofdstuk ter sprake zal komen, wordt bij voorkeur gebruikt voor de opslag van (kleine) deelverzamelingen van een bestand.

Om een record, gegeven zijn sleutelwaarde, te vinden zal zowel bij consecutieve opslag als bij opslag in een lijststructuur een zoekproces nodig zijn. Er is immers in deze gevallen geen (direct) verband tussen sleutelwaarde en adres. Voor dit zoeken zijn diverse technieken ontwikkeld, die in een apart hoofdstuk ter sprake zullen komen.

Om bestanden efficiënt te kunnen verwerken, zal het wel eens nodig zijn de records in een andere volgorde te plaatsen. Dit geldt met name voor mutatiebestanden. Om deze andere volgorde te krijgen zal het bestand volgens het nieuwe volgordecriterium moeten worden gesorteerd. Naast het zoek- is er dus ook een sorteerprobleem, waarop ook later zal worden ingegaan.

Het directe verband tussen sleutelwaarde en adres, waarvan sprake is bij de directe bestandsorganisatie, is over het algemeen niet eenvoudig te effectueren. Er zal dan ook de nodige aandacht aan de conversie van sleutel naar adres worden besteed.

In het voorgaande is de sleutelwaarde van een record gebruikt als criterium voor de benadering van een record. Er zullen echter soms ook niet-sleutelwaarden worden gebruikt als benaderingscriterium. Zo kan bijv. worden gevraagd van een studentenbestand een lijst af te drukken van alle studenten, die in Eindhoven wonen en Athenaeum B vooropleiding hebben. Voor de beantwoording van deze vraag is dan niet relevant de waarde van het sleutelattribuut registratienummer, maar de waarden van de attributen woonplaats en soort vooropleiding. Het is dan zaak op grond van de waarden 'Eindhoven' resp. 'Athen B' de records op te sporen die deze waarden bevatten. Zo'n opsporing wordt meestal met de (Engelse) term *retrieval* aangegeven. Hiervoor zullen in het algemeen zogeheten geïnverteerde bestanden (*inverted files*) nodig zijn. Deze komen in een apart hoofdstuk aan bod.

Ofschoon een beschouwing over zogeheten *databases* eigenlijk buiten het bestek van een boek over bestandsorganisatie valt, is hierover toch aan het einde van dit boek een inleidend hoofdstuk opgenomen. Mede hierdoor is het hopelijk mogelijk enig inzicht te hebben in de plaats, die bestandsorganisatie inneemt in (de implementatie van) databases.

Samenvattend zal dus verder ter sprake komen:

Sequentiële bestandsorganisatie

Lijststructuren

Zoekmethoden

Sortering
Directe bestandsorganisatie
Index-sequentiele organisatie
Index-directe organisatie
Retrieval; geïnveteerde bestanden
Databases

OPGAVEN

- 2.1.
 - a. Noem nog enkele volgens u relevante objecten bij de organisaties van voorbeeld 2.1.1.
 - b. Noem enkele volgens u relevante objecten voor de volgende twee organisaties:
 - autoverhuurbedrijf
 - postorderbedrijf
- 2.2. Voor een handelsonderneming moet per artikel worden vastgelegd:
 - artikelnummer
 - artikelomschrijving
 - leveranciers (onbepaald aantal leveranciers)
 - per leverancier:
 - . code leverancier
 - . naam, adres, woonplaats, telefoon
 - . gemiddelde levertijd (onafhankelijk van artikel)
 - . van alle afleveringen van afgelopen jaar: het afgeleverde aantal en het faktuurbedrag per aflevering
 - verkopen van de laatste 12 maanden: per maand aantal verkochte exemplaren en verkoopbedrag.
 - a. Leg de logische structuur vast in één objecttype.
 - b. Leg de logische structuur vast met meer dan één objecttype.
- 2.3. Bedenk een occurrence 3 bij voorbeeld 2.2.1.
- 2.4. (zie voorbeeld 2.1.1).
 - a. Geef (volgens u) relevante attributen van de bij onderwijsinstelling genoemde objecten.
 - b. Geef vervolgens van elk objecttype twee occurrences.
 - c. Kunnen alle sub a genoemde attributen in één objecttype voor onderwijsinstelling worden ondergebracht? Zo ja, geef de beschrijving van dit objecttype.
- 2.5. In voorbeeld 2.1.1 is sprake van een productiebedrijf. Veronderstel dat voor elk van de genoemde objecttypen een bestand is opgezet. Beredeneer voor elk van deze bestanden hoe hoog

(in de schaal: niets, weinig, matig, groot, zeer groot) naar uw
schatting de bestandsaktiviteit, -verandering, -verloop en
-groei zullen zijn.

3 SEQUENTIELE BESTANDSORGANISATIE

3.1 INLEIDING

Een bestand is sequentieel georganiseerd als de records in (meestal oplopende) sleutelvolgorde zijn opgeslagen en in deze volgorde toegankelijk zijn. Deze organisatie is betrekkelijk eenvoudig, wat o. a. tot uitdrukking komt in de geringe eisen die het stelt aan de opslagstructuur en de fysieke opslagmedia. Ze is dan ook het langste in gebruik en wordt ook nu nog veel toegepast. Deze eenvoud heeft natuurlijk ook haar keerzijde en wel dat men gebonden is aan sequentiële toegang.

Om bij sequentiële doorloop van een bestand het lezen van onnodige gegevens te vermijden, kan het overweging verdienen in plaats van één, meer dan één objekttype (en dus ook meer dan één bestand) te gebruiken ('splitsen' van bestanden).

VOORBEELD 3.1

Het objekttype student (zie beschrijving (6) in paragraaf 2.1) zou kunnen worden gesplitst in een objekttype met (rnr, naw(nm, adr, wpl)) en een objekttype met rnr en de overige attributen. In vele toepassingen immers heeft men of naw niet nodig of uitsluitend nodig.

In principe is men bij de sequentiële organisatie geheel vrij in de keuze van recordstructuur. Men kan dus vaste of variabele lengte records gebruiken met daarbij de verschillende varianten, zoals beschreven in paragraaf 2.3. In de praktijk maakt men liefst gebruik van vaste lengte records.

De records van een sequentieel georganiseerd bestand worden vrijwel altijd consecutief opgeslagen. Mogelijke andere opslagstructuren zullen wij in het volgende hoofdstuk bespreken. In dit hoofdstuk zullen wij uitsluitend met consecutieve opslag werken.

In principe komt zowel band als schijf als geschikt fysiek geheugenmedium in aanmerking. Dikwijls wordt de voorkeur gegeven aan magneetband (lage kosten, eenvoudige verwerking). Hierbij moet wel worden bedacht dat een magneetband een niet-adresseerbaar

medium is, zodat records niet lokaal veranderd kunnen worden. Dit betekent dat bij elke verandering in het bestand, dit bestand in zijn geheel (behalve de eventuele weglatingen) moet worden herschreven (zie ook appendix I). Het oude bestand kan voor reconstructiedoeleinden gebruikt worden (grootvader-vader-zoon systeem).

Het herschrijven van een sequentieel georganiseerd bestand is bij uitsluitend veranderingen niet nodig als het bestand met vaste lengte records is opgeslagen op een adresseerbaar medium (bijv. schijf).

Om de gemiddelde 'insteltijd' per record zo klein mogelijk te maken en tevens de bezettingsgraad van het achtergrondgeheugen zo hoog mogelijk op te voeren, worden de records dikwijls in blokken op een geheugenmedium opgeslagen. Hier werd reeds op gewezen in hoofdstuk 1, par. 1.3.

3.2 ONDERHOUD; BESTANDSMUTATIES

Het plegen van onderhoud op een sequentieel georganiseerd bestand is in principe eenvoudig voor ieder van de drie genoemde mutatiesoorten afzonderlijk.

- Invoegen : plaats na het record, dat op grond van de sleutelwaarde onmiddellijk voorafgaat aan het in te voegen record dit nieuwe record in het (noodzakelijk herschreven) bestand.
- Veranderen : voer op het desbetreffende record de gewenste verandering uit en plaats het veranderde record in het (zo nodig herschreven) bestand.
- Verwijderen : 'verwijder' het desbetreffende record door dit:
- bij herschrijving van het gehele bestand niet in het nieuwe bestand op te nemen,
 - voorzien van een 'verwijderingskenmerk' in het nieuwe bestand op te nemen, indien het bestand niet in zijn geheel wordt herschreven.

De sequentiële organisatie heeft echter enkele consequenties voor het efficiënt uitvoeren van dit onderhoud. Immers bij deze organisatie kan een bepaald record slechts door zoeken in het bestand worden gevonden. Hierbij maakt het wel enig verschil uit of het bestand op een adresseerbaar dan wel een niet-adresseerbaar medium ligt opgeslagen. We zullen in deze paragraaf uitgaan van opslag op een niet-adresseerbaar medium (een magneetband). Voor gebruik van adresseerbare media zij verwezen naar paragraaf 3.3, de opgaven en naar hoofdstuk 4.

a. Voor het zoeken van één willekeurig record in een sequentieel bestand zal gemiddeld het halve bestand moeten worden gelezen. Bij grote bestanden van bijv. enkele tienduizenden records en een lengte van rond honderd tekens per record, gaat dit verhoudingsgewijs een 'zee van tijd' kosten. (Opgaven 1.5 en 1.6 hebben reeds duidelijk gemaakt dat we dan in de orde van grootte van enkele minuten aan zoeken moeten besteden, hetgeen in geen verhouding staat tot de processortijd.)

b. Onderhoud van een sequentieel bestand met een lage bestandsactiviteit (waarbij dus relatief slechts een gering aantal records van het bestand is betrokken) leidt dus tot een onaanvaardbaar scheve verhouding tussen I/O- en processortijd. Voor een 'balanced run' zal de bestandsactiviteit hoog moeten zijn, d.w.z. een groot aantal mutaties (transactions) moet worden gepleegd. Een daling van de gemiddelde zoektijd per stamrecord door toename van de bestandsactiviteit is uiteraard alleen te realiseren als ook het mutatiebestand vooraf is gesorteerd op dezelfde sleutel als het permanente bestand.

c. Om een hoge bestandsactiviteit te bereiken zal het nodig zijn de mutaties gedurende enige tijd 'op te sparen'. Dit heeft wel tot gevolg dat het stambestand alleen vlak na een verwerkingsgang de feitelijke situatie precies weergeeft.

d. Om een onaanvaardbare zoektijd te vermijden, zal men dus bij de sequentiële organisatie twee nadelige gevolgen voor lief moeten nemen, nl. de onder b genoemde sorteergang van het mutatiebestand en het onder c gesignaleerde niet altijd bijgewerkt zijn van het stambestand. In vele gevallen zullen de voordelen van de betrekkelijke eenvoud en (daardoor) relatief geringe kosten van een sequentiële organisatie ruimschoots opwegen tegen genoemde nadelen. Er zijn echter gevallen, waar dit niet opgaat. Zo zal bijv. bij een reserveringssysteem voor een luchtvaartmaatschappij de bezetting van vliegtuigen altijd precies bekend moeten zijn. In dergelijke gevallen is de conclusie zonder meer, dat de sequentiële bestandsorganisatie ongeschikt is.

Samenvattend kunnen we stellen dat sequentiële bestandsorganisatie alleen zinvol is als sprake is van een hoge bestandsactiviteit met gebruik van gesorteerde mutatiebestanden, waarbij voor het informatiesysteem het niet altijd bijgewerkt zijn van het stambestand toelaatbaar is.

We zullen nu de methodiek voor het onderhoud van sequentiële bestanden behandelen en wel in twee stappen. In stap A zal uitsluitend met één mutatiesoort worden gewerkt en wel verandering (updating). Voor verwerkingsgangen met enkel invoegen of enkel verwijderen zij

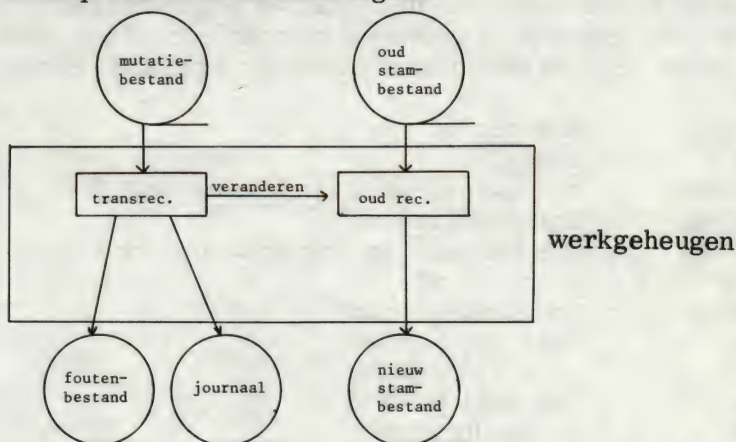
verwezen naar de opgaven. In stap B wordt de combinatie van de drie mutatiesoorten tegelijk behandeld.

A. ONDERHOUD MET ALLEEN VERANDERING (UPDATING)

Om de methodiek concreet gestalte te kunnen geven, maken we de volgende veronderstellingen:

- A.1. Verandering van een bestaand record is mogelijk t.a.v. alle velden behalve de sleutel. Sleutelverandering wordt achteraf apart behandeld.
- A.2. Een bepaalde waarde van de sleutel kan hoogstens in één record van het stambestand voorkomen, en in nul, één of meer mutatierecords. Mutaties op eenzelfde record uit het stambestand mogen in willekeurige volgorde worden uitgevoerd.
- A.3. Het stambestand en het mutatiebestand zijn op niet-dalende sleutelwaarden gesorteerd.
- A.4. In de praktijk kunnen allerlei (door vergissingen geïntroduceerde) fouten optreden. Hier zullen we uitsluitend rekening houden met de volgende mogelijke fout: een sleutelwaarde in het mutatiebestand komt niet voor in het stambestand.
- A.5. Van goede mutaties moet een verslag (journaal of audit trail) op magneetband worden geproduceerd, bestaande uit een kopie van de desbetreffende mutaties. Foute mutaties moeten niet worden aangebracht maar worden gesignaleerd in een 'foutenbestand' (op magneetband), eveneens bestaande uit een kopie van de desbetreffende mutaties.
- A.6. Het afdrukken van de banden met journaal en foutenlijst zal verder buiten beschouwing blijven.

Voor dit mutatieprobleem is eenvoudig een programma(schets) te ontwerpen met behulp van onderstaande figuur.



Invoerbestanden voor de gevraagde verwerking zijn het oude stambestand en het mutatiebestand (beiden gesorteerd op sleutelwaarde). Uitvoerbestanden zijn het nieuwe stambestand, het journaalbestand en het foutenbestand.

De recordgebieden transrec resp. oudrec in het werkgeheugen fungeren als inleesgebieden voor het mutatiebestand resp. oudstambestand en als uitleesgebieden voor journaal en foutenbestand resp. nieuw stambestand.

De kern van het proces dat de gevraagde verwerking tot stand brengt zal moeten bestaan uit een of meer goed gekozen herhalingsopdrachten. Bij bestandsbewerkingen is het belangrijk de 'draagwijdte' van de herhalingsopdracht goed te kiezen. In het onderhavige voorbeeld zullen wij per 'doorgang' van de herhalingsopdracht het volgende uitvoeren: de verwerking van de laagste van de nog niet verwerkte sleutelwaarden uit het oude stambestand en het mutatiebestand. Dit leidt tot de volgende operaties in de herhalingsopdracht. Is de sleutelwaarde van het (laatst gelezen) stamrecord kleiner dan of gelijk aan de sleutelwaarde van het (laatst gelezen) transactierecord,

zo ja : dan alle transactierecords met eventueel dezelfde sleutelwaarde als het stamrecord gebruiken om het stamrecord te wijzigen, het stamrecord wegschrijven naar het nieuwe stambestand en het volgende record uit het oude stambestand inlezen.

zo nee : dan alle transactierecords met dezelfde sleutelwaarde als die van het laatst gelezen transactierecord naar het foutenbestand schrijven.

Voorafgaand aan de uitvoering van de herhalingsopdracht is uiteraard een initialisatie nodig in de vorm van het lezen van het eerste record in het stambestand en in het mutatiebestand.

Na het bovenstaande zal de bijgaande programmaschets duidelijk zijn (voor de gebruikte taalelementen raadplege men zo nodig appendix II, voor de gekozen namen van variabelen bijgaande lijst).

strec	: <u>stam</u> record
sl	: <u>sleutel</u>
stdata	: overige gegevens van <u>stam</u> record
trrec	: <u>transactie</u> record
trdata	: overige gegevens van <u>transactie</u> record
stb	: <u>stam</u> bestand
trb	: <u>transactie</u> bestand
ob	: <u>oude</u> <u>stam</u> bestand
nb	: <u>nieuwe</u> <u>stam</u> bestand
tb	: <u>transactie</u> bestand
jb	: <u>journaal</u> bestand
fb	: <u>fouten</u> bestand


```

or      : oud record (variabele)
tr      : transactierecord (variabele)
hsl     : hulpsleutel (variabele)

```

```

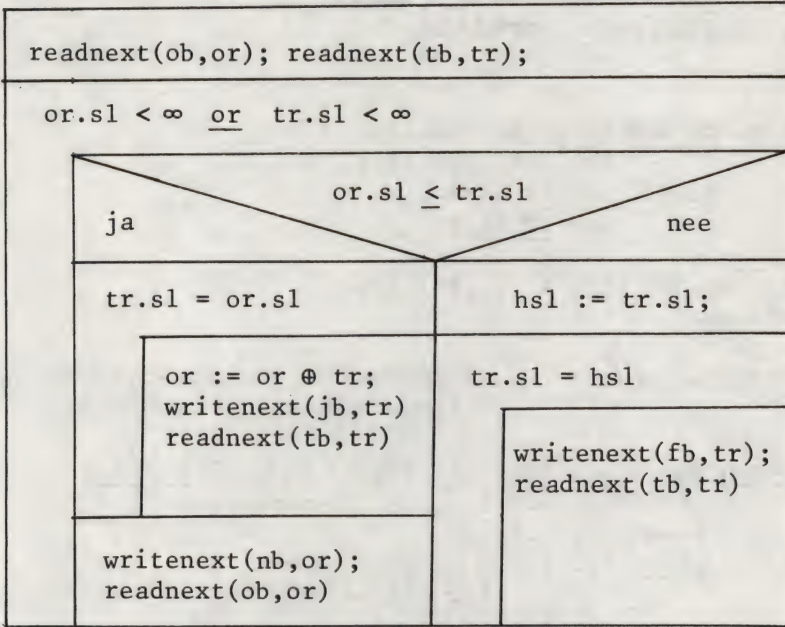
begin type record strec(sl,stdata);
                                trrec(sl,trdata);
                                file   stb of strec,
                                trb of trrec
                                endtype;
var ob,nb: stb; tb,jb,fb: trb
endvar;

ready(ob,tb);                  %stambestand en transactiebestand
                                beschikbaar voor verwerking%

begin var or: strec; tr: trrec; hsl: sl endvar;
    readnext(ob,or);           %initialisatie%
    readnext(tb,tr);
    while or.sl <  $\infty$  or tr.sl <  $\infty$ 
        do                      %laagste sleutelwaarde verwerken%
            if or.sl < tr.sl
                then %transacties eventueel verwerken%
                    while tr.sl = or.sl
                        do or := or  $\oplus$  tr;
                        writenext(jb,tr);
                        readnext(tb,tr)
                        od;
                    writenext(nb,or);
                    readnext(ob,or)
                else %or.sl > tr.sl, dus onverwerk-
                    bare transactie%
                    hsl := tr.sl;
                    while tr.sl = hsl
                        do writenext(fb,tr);
                        readnext(tb,tr)
                        od
                fi
            od
    end;
    save(nb,jb,fb)             %gevraagde output zeker stellen%
end

```

Een corresponderend structuurdiagram ziet er als volgt uit.



B. ONDERHOUD MET DRIE MUTATIESOORTEN TEGELIJK

Bij het onderhoud van een bestand zullen in het algemeen alle drie mutatiesoorten (invoegen, veranderen, verwijderen) aan bod komen. Een voor de hand liggende en in de praktijk nog vaak gebruikte methodiek is dan voor elke mutatiesoort een aparte verwerkingsgang te laten plaatsvinden. Deze methodiek moet echter worden verworpen ten gunste van een methodiek waarbij alle mutaties in één gang worden verwerkt. Bij aparte verwerking per mutatiesoort zijn de nadelen:

- het stambestand wordt drie keer gelezen en opnieuw geschreven, waarmee veelal de totale verwerkingstijd onnodig wordt verdriedvoudigd;
- het vereist grote oplettendheid van de operators, omdat de mutaties beslist in de goede volgorde (invoegen, veranderen, verwijderen) moeten worden uitgevoerd (ga dit na!);
- foutmeldingen en journalen, betrekking hebbende op eenzelfde record staan over meer staten verspreid, hetgeen het controleren van deze staten bemoeilijkt.

Dat veelal toch de omslachtige methodiek van aparte verwerkingsgangen wordt gebruikt is vermoedelijk te wijten aan het feit dat het in één verwerkingsgang aanbrengen van alle mutaties tegelijk moei-

lijk wordt gevonden. Hierbij komt de vrees (niet ten onrechte) fouten te maken in het programma.

Bij verwerking in één gang zal in de mutatierecords (transacties) moeten worden aangegeven om welke mutatiesoort het gaat. Hiervoor zal in het mutatierecord een veld, trcode, worden opgenomen. Door de waarde van trcode zal worden aangegeven of men met een invoeging, verandering of verwijdering te maken heeft. Bij eenzelfde sleutelwaarde dient wederom de volgorde invoegen, veranderen, verwijderen te worden aangehouden. Geschikte waarden voor trcode zouden bijvoorbeeld zijn: 10 (invoegen), 20 (veranderen), 30 (verwijderen).

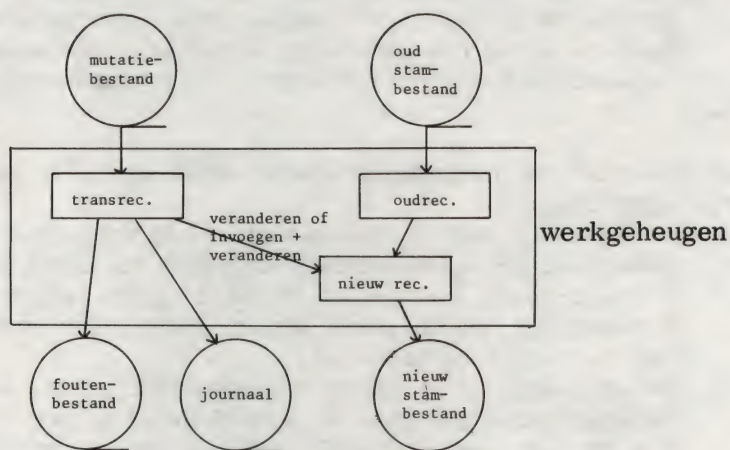
Voor een concrete programmaschets maken we ook nu weer enkele veronderstellingen:

- B.1. Een bepaalde waarde van de sleutel kan hoogstens in één record van het stambestand voorkomen en in nul, één of meer mutatierecords, echter wel zo dat invoegen hoogstens één maal, veranderen 0, een of meer malen, verwijderen hoogstens één maal behoort voor te komen.
- B.2. Het stambestand en het mutatiebestand zijn naar niet-dalende sleutelwaarden gesorteerd, terwijl in het mutatiebestand records met gelijke sleutel in niet-dalende volgorde van trcode staan.
- B.3.
 - invoegen van een record geschiedt door het op de 'juiste plaats' in het nieuwe stambestand te zetten;
 - verwijderen van een record geschiedt door het niet op te nemen in het nieuwe stambestand;
 - verandering van een bestaand record is mogelijk voor alle velden behalve de sleutel.
- B.4. Men moet rekening houden met de volgende mogelijk optredende fouten:
 - invoegen van een record met de sleutelwaarde van een bestaand record;
 - veranderen of verwijderen op grond van een sleutelwaarde, die niet voorkomt in het stambestand of bij de in te voegen records;
 - onbekende mutatiecode (dus geen invoeging, verandering of verwijdering).
- B.5. Journaal en foutenband als in A.5.
- B.6. Als A.6.

Ten aanzien van B.4 (eventuele fouten) moet nog worden opgemerkt, dat het ook mogelijk is dat de sorteervolgorde van stambestand of mutatiebestand op een of meer punten is verstoord. Dit is voor de sequentiële verwerking uiteraard funest en zou resulteren in legio foutmeldingen (ga dit na!). In het kader van dit boek zullen wij

hieraan echter geen aandacht besteden, omdat een controle van de sorteervolgorde gemakkelijk uit te voeren is door bij iedere leesoperatie de sleutel van het laatst gelezen record te vergelijken met die van het vorige record.

Voor de programmering van het mutatieproces gaan we uit van het voor de hand liggende schema, dat hieronder is gegeven.



In- en uitvoerbestanden zijn dezelfde als in geval A. In het werkgeheugen is echter een uitleesgebied (nieuwrec) bijgekomen, nodig omdat op een in te voegen record ook nog veranderingen kunnen volgen. Het in te voegen record kan niet op de plaats van oudrec worden gezet, aangezien dan het laatst ingelezen record van het oude stambestand verloren zou gaan. Evenmin kan men het in te voegen record in transrec laten staan, want dan zou het worden overschreven door het eerstvolgende record uit het mutatiebestand.

Ook in dit geval zal de kern van het verwerkingsproces bestaan uit een goed gekozen herhalingsopdracht. De keuze voor de 'draagwijdte' van deze herhalingsopdracht is dezelfde als die bij A, dus verwerking van de laagste van de nog te verwerken sleutelwaarden uit het oude stambestand en het mutatiebestand. Het verschil met A zit hem uiteraard in de andere uitwerking van de herhalingsopdracht:

- bepaal de laagste sleutelwaarde van de nog niet verwerkte sleutelwaarden
- komt deze laagste waarde in het stambestand voor?
 zo ja : dan oudrec naar nieuwrec, volgende stamrecord lezen, en te kennen geven dat nieuwrec een kandidaat bevat voor het nieuwe stambestand;

zo nee : is transactie een invoeging?

zo ja : dan transrec naar nieuwrec; journaal bijwerken, te kennen geven dat nieuwrec een kandidaat bevat voor het nieuwe stambestand;

zo nee: foutenbestand bijwerken, te kennen geven dat nieuwrec nog geen kandidaat bevat voor het nieuwe stambestand,

tenslotte volgende transactie inlezen.

- alle transactierecords verwerken die dezelfde sleutelwaarde hebben als de geldende laagste sleutelwaarde. Deze herhalingsopdracht houdt in: verwerk het aan de beurt zijnde transactierecord. Door een verwijderingsopdracht zal de kandidatuur van nieuwrec voor het nieuwe stambestand worden ingetrokken.
- Is nieuwrec (nog) kandidaat dan nieuwrec wegschrijven naar nieuw stambestand.

De initialisatie is uiteraard als in geval A.

Hieronder volgt nu de programmaschets (p.46).

Opmerkingen:

- met de boolean new wordt de kandidatuur van nieuwrec voor het nieuwe stambestand weergegeven;
- in regel (x) kan (nr.sl := tr.sl; nr.data := tr.data) niet vervangen worden door nr := tr, want nr en tr zijn niet van hetzelfde type.
- aan de records in de foutenband is niet te zien om welke fout het ging.

Het corresponderende structuurdiagram is gegeven op p.47.

Tenslotte zullen we het (uitgestelde) probleem van sleutelverandering behandelen. Sleutelverandering vormt een apart probleem omdat hierbij de sequentiële volgorde van het stambestand kan worden verstoord. Een op sleutelwaarde gewijzigd record kan dus niet zonder meer op het nieuwe stambestand worden weggeschreven. Afhankelijk onder meer van het aantal sleutelveranderingen kan een keus worden gemaakt uit de volgende behandelingsmogelijkheden.

1. Neem twee mutatierecords in het mutatiebestand op, één met de oude sleutel om het oude record te verwijderen en één met de nieuwe sleutel om een compleet nieuw record toe te voegen. De mutatierecords met oude resp. nieuwe sleutels moeten een verwijdering resp. invoeging tot gevolg hebben. Dit betekent in feite dat het mutatierecord met de nieuwe sleutel ook alle gegevens, behalve de sleutelwaarde van het oude record moet bevatten.

2. Neem een sleutelmutatierecord in het mutatiebestand op. Met behulp van dit record wordt dan de sleutelwaarde van het oude record

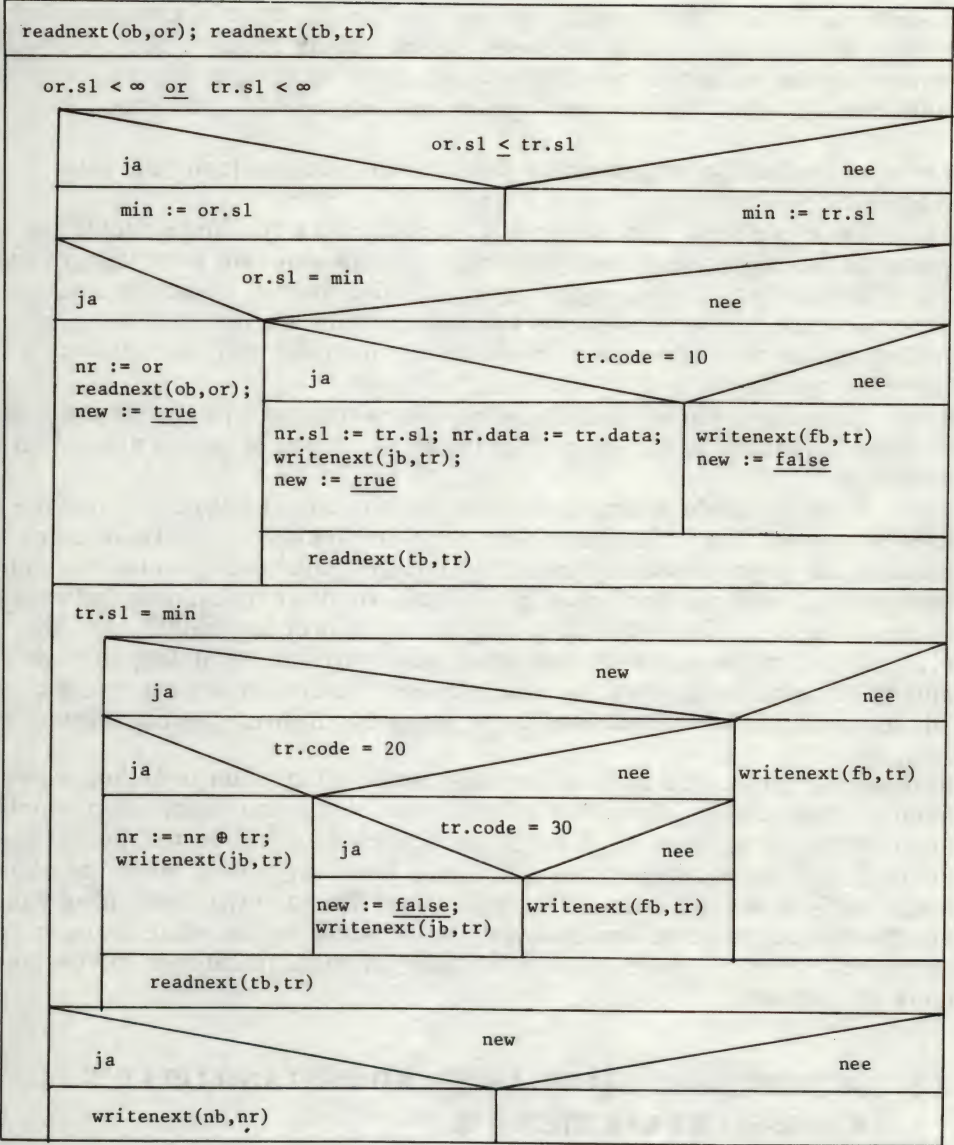
```

begin type record strec(sl,data),trrec(sl,code,data);
      file stb of strec, trb of trrec
endtype;
var ob,nb: stb; tb,jb,fb: trb
endvar;

ready(ob,tb);           %input bestanden beschikbaar%

begin var or,nr: strec; tr: trrec; min: sl; new: boolean
endvar;
readnext(ob,or); readnext(tb,tr);
while or.sl < ∞ or tr.sl < ∞
do if or.sl < tr.sl then min := or.sl else min := tr.sl fi;
  if or.sl = min
  then nr := or; readnext(ob,or); new := true
  else if tr.code = 10 %10: invoeging%
  then nr.sl := tr.sl; nr.data := tr.data;
    writenext(jb,tr);
    new := true
  else writenext(fb,tr); new := false
  fi;
  readnext(tb,tr)
fi;
while tr.sl = min
do if new
  then if tr.code = 20 %20: verandering%
  then nr := nr ⊕ tr; writenext(jb,tr)
  else if tr.code = 30 %30: verwijdering%
  then new := false;
    writenext(jb,tr)
  else %onbekende trcode%
    writenext(fb,tr)
  fi
  else %transactie niet te plaatsen%
    writenext(fb,tr)
  fi;
  readnext(tb,tr)
od;
if new then writenext(nb,nr) fi
od
end;
save(nb,jb,fb)
end

```

(op het moment dat het volgens de bestaande sleutel 'aan de beurt' is) gewijzigd. Dit gewijzigde record wordt naar het nieuwe stambestand weggeschreven. Na de verwerking van alle mutaties moet dan op het nieuwe stambestand een aparte sorteergang plaatsvinden (zie hoofdstuk 5).

3. Als sub 2 met dit verschil, dat het gewijzigde record niet naar het nieuwe stambestand wordt weggeschreven, maar 'opgeslagen' voor opname in het volgende mutatiebestand als in te voegen record. Het oude record wordt tijdens het lopende mutatieproces verwijderd.

Ten aanzien van bovengenoemde drie behandelingswijzen valt het volgende op te merken.

Ad 1. Deze methode lijkt eenvoudig en aantrekkelijk, maar heeft het grote (meestal onoverkomelijke) bezwaar dat dan alle recordgegevens die in het bestand liggen opgeslagen, ook daarbuiten bekend moeten zijn. Hooguit zal deze methode toepasbaar zijn als het gaat om een gering aantal sleutelveranderingen op een bestand met records van beperkte omvang.

Ad 2. Deze methode is uit efficiency-overwegingen in het algemeen af te raden, behalve in het (uitzonderlijke) geval van massale sleutelverandering.

Ad 3. Deze methode wordt in de praktijk het meest gebruikt. Het nadeel is wel dat het stambestand na een verwerkingsgang alleen compleet is als geen sleutelverandering in deze gang heeft plaatsgevonden. Indien dit nadeel als een ernstig bezwaar wordt gevoeld, kan in bepaalde gevallen deze derde methode nog als volgt worden 'verfijnd'. Als nl. het gewijzigde record 'verderop' moet worden geplaatst, kan dit record na een 'verblijf in een wachtkamer' later worden ingevoegd. Dit leidt uiteraard wel tot een meer ingewikkeld mutatieprogramma.

In deze paragraaf 3.2 hebben we uitsluitend gesproken over het onderhoud van bestanden. Over het gebruik van bestanden volgt geen aparte bespreking, aangezien de daarbij optredende problemen van bestandsverwerking van dezelfde aard zijn als in het voorgaande en in het algemeen zelfs maar ten dele zullen optreden. Zo zal bij raadpleging van een bestand dit niet op een nieuw volume (band) behoeven te worden weggeschreven. Verder zij voor enkele gebruiksproblemen verwezen naar de opgaven.

3.3 SEQUENTIEEL GEORGANISEERD BESTAND OP EEN ADRESSEERBAAR MEDIUM

In de vorige paragraaf hebben we ons beperkt tot de behandeling van bestanden, die op magneetbanden (dus een niet-adresseerbaar) medium liggen opgeslagen. In paragraaf 3.1 werd er echter reeds op gewezen

dat een sequentieel bestand ook op een adresseerbaar medium, bijv. schijf, kan worden opgeslagen. In dit geval hoeft bij een mutatieproces, dat alleen veranderingen op records van vaste lengte inhoudt, geen nieuw stambestand te worden gecreëerd.

Door de adresseerbaarheid van het medium is het in principe mogelijk dat elke willekeurige recordplaats wordt gelezen en dan overschreven. (We laten hierbij buiten beschouwing dat in sommige implementaties van de sequentiële organisatie een dergelijke benadering niet is toegestaan.) Wordt gevraagd naar het duizendste record, dan kan direct bij het beginadres van het bestand 999 maal de recordlengte worden opgeteld om zodoende het adres van dat duizendste record te verkrijgen. Door de adresseerbaarheid van het medium kan dan het lees/schrijfmechanisme direct op dat adres worden ingesteld. Uiteraard zal opslag op een adresseerbaar medium leiden tot wijzigingen in de programmatuur voor onderhoud en gebruik. Hiervoor zij verder verwezen naar de opgaven.

Moeilijker is het als bij mutaties niet alleen sprake is van veranderingen, maar ook van invoegingen en/of verwijderingen. Immers, zou men, ook op het adresseerbare geheugenmedium, de sequentiële ordening der records uitsluitend via consecutieve opslag willen weergeven, dan zou invoeging en/of verwijdering moeten leiden tot een herschrijving van het gehele bestand. Intuïtief wordt dit, bij een adresseerbaar medium, (terecht) ervaren als 'zonde van de tijd, die het kost'. Daarom zal het meestal zinvol zijn 'enigszins' af te wijken van de consecutieve opslag. Dit kan volgens verschillende varianten. Een variant zullen we nu globaal beschrijven.

Bij de creatie van het bestand worden de records consecutief opgeslagen. Een record heeft een speciaal (boolean) veld, dikwijls flag genoemd. Deze flag kan twee waarden hebben. De ene waarde (bijvoorbeeld 0) geeft aan dat het gaat om een bestaand record, en de andere waarde (bijvoorbeeld -1) geeft aan dat het gaat om een in feite verwijderd record. Een verwijdering vindt dus plaats door de flag van het desbetreffende record de waarde -1 te geven. Invoeging is als volgt te behandelen. Wordt een record ingevoegd, dan wordt het aan het eind van het bestand opgeslagen, dus na het oude laatste record. In het record, waarop het qua sleutelwaarde moet volgen, wordt een verwijzing opgenomen naar het adres van het ingevoegde record. Zodoende blijft het toch mogelijk bij het gebruik van het bestand dit sequentieel te benaderen. De opslagstructuur die daarvoor nodig is, is een zogeheten lijststructuur. Deze zullen wij in het volgende hoofdstuk bespreken, daar zij ook bij andere vormen van bestandsorganisatie veelvuldig gebruikt wordt.

3.4 APARTE PROBLEMEN BIJ VERWERKING

We besluiten dit hoofdstuk over sequentiële bestandsorganisatie met enkele opmerkingen.

1. Over multi-volume files. Wanneer een sequentieel bestand over meer informatiedragers is gesplitst (bijv. op grond van activiteitsoverwegingen) dan wordt de programmatuur voor onderhoud en gebruik ingewikkelder. Op dit probleem wordt hier verder niet ingegaan.

2. Over afbreken en herstarten van een mutatieproces. Hierbij moet men zeer voorzichtig zijn wat de veranderingen betreft. Is de verandering een vervanging van één of meer oude velden door nieuwe velden, dan kan er weinig kwaads gebeuren bij een tussentijds afbreken van een mutatieproces. Is de verandering daarentegen een vermeerdering of vermindering van één of meer recordvelden, dan kan het bij een tussentijds afbreken en later weer herstarten van een mutatieproces gebeuren dat de mutatie ten onrechte nul of twee keer wordt uitgevoerd! Men kan er immers nooit op rekenen dat de operator weet tot welk punt de mutaties met succes zijn aangebracht. Men dient er daarom bij een verandering voor te zorgen:

- of alleen maar vervanging toe te staan (denk aan adresveranderingen),
- of als dit door de aard van een probleem niet mogelijk is (denk aan magazijnmutaties), de uitvoering van een verandering afhankelijk te maken van een tweede kenmerk (naast de transactiecode) in het transactierecord en dit over te nemen in het nieuwe bestand. De verandering mag alleen dan aangebracht worden, wanneer dit tweede kenmerk in transactierecord en bestandsrecord niet overeenstemmen! Voor dit tweede kenmerk kan afhankelijk van de aard van het probleem bijvoorbeeld een datum of een bonnummer, enz. gebruikt worden. Op dit herstartprobleem wordt in dit boek ook niet nader ingegaan.

OPGAVEN

- 3.1. Maak een programmaschets voor onderhoud van een sequentieel bestand, waarbij uitsluitend invoeging plaatsvindt en verder, mutatis mutandis, de veronderstellingen A.1 tot en met A.6 (par. 3.2) gelden.
- 3.2. Als 3.1 maar nu in geval van enkel verwijderingen.
- 3.3. Gegeven is een sequentieel personeelsbestand van een instelling. De recordindeling per personeelslid is als volgt:

veld

1. registratienummer
2. naam
3. adres
4. woonplaats
5. datum-in-dienst-treding
6. burgerlijke staat
7. aantal kinderen
8. maandsalaris (in guldens)

Het registratienummer is samengesteld uit:
geboortedatum
volgnummer
controlecijfers

Het volgnummer wordt bepaald door een nieuw personeelslid een volgnummer te geven dat 1 hoger is dan het hoogste reeds aanwezige volgnummer bij dezelfde geboortedatum.

De twee controlecijfers worden bepaald door het zevencijferig getal van geboortedatum + volgnummer te delen door 13 en de rest van deze deling als controlegetal op de 8e en 9e positie te plaatsen.

Op het bestand vinden periodiek mutaties plaats, en wel:

- invoegen van nieuwe personeelsleden
 - veranderingen op bestaande records, met name op één of meer van de items 3, 4, 6, 7 en 8.
 - verwijderingen van personeelsleden wegens ontslagname, enz.
- Deze drie mutatiesoorten worden in één gang verwerkt.

Gevraagd: een recordindeling voor het mutatiebestand.

- 3.4. Gegeven zijn twee sequentieel georganiseerde bestanden B-1 en B-2. Beide bestanden hebben dezelfde recordindeling (met sleutelveld S-1 resp. S-2). Een waarde van de sleutel kan niet in beide bestanden tegelijk voorkomen. Elk bestand bevat minstens één record.

Gevraagd: een programmaschets voor de creatie van een sequentieel georganiseerd bestand B-3, dat een samenvoeging (merge) is van B-1 en B-2.

- 3.5. a. Is een sequentieel bestand op een adresseerbaar medium in feite niet een bestand met een directe organisatie?
- b. Waarom speelt file-turnover geen rol bij een sequentieel bestand op magneetband en waarom kan het wel een rol spelen bij opslag op schijf?
- c. Voor het zoeken van één willekeurig record in een sequentieel bestand op magneetband moet gemiddeld de helft van het

totaal aantal records worden gelezen. Geldt hetzelfde als het bestand op een adresseerbaar medium ligt opgeslagen?

- d. Als een ongeblokt sequentieel stambestand van 1000 records op magneetband wordt gemuteerd met behulp van een ongeblokt mutatiebestand van 600 records op magneetband, hoeveel lees- en schrijfp opdrachten zijn dan nodig?

- 3.6. Gegeven een klantenbestand met 5000 klantenrecords en een artikelenbestand met 100 artikelrecords. Beide bestanden zijn sequentieel georganiseerd op magneetband. Een klantenrecord bevat naast allerlei andere velden o. a. een vector van 100 elementen. Het i^e element van deze vector bevat de omzet in aantallen van het i^e artikel uit het artikelbestand. Een artikelrecord bevat o. a. de prijs per stuk van het desbetreffende artikel. Ga na hoe (zo efficiënt mogelijk) het totale omzetbedrag per klant kan worden bepaald.

Opmerking: Het bovenstaande is een concreet geval van het volgende, meer abstract geformuleerde probleem:

Gegeven een bestand A met n records a_i en een bestand B met m records b_j . Beide bestanden zijn sequentieel georganiseerd op magneetband. Elk record a_i bevat o. a. een vector, bestaande uit m elementen a_{ij} (dus $1 \leq j \leq m$). Elk record b_j bevat een veld b_{jk} . Ga na hoe (zo efficiënt mogelijk)

$$\sum_{j=1}^m a_{ij}b_{jk} \text{ voor elke } i \text{ kan worden berekend.}$$

- 3.7. Geef aan welke wijzigingen moeten worden aangebracht in het schema en het programma, behorende bij geval A, als sprake is van een mutatieproces, uitsluitend bestaande uit veranderingen, op een sequentieel bestand op schijf.
- 3.8. Voor een tijdschrift-abonnementenadministratie moet een systeem worden opgezet met de volgende (zeer beperkte) doelstellingen:
1. halfjaarlijks moet een lijst kunnen worden afgedrukt van alle op dat moment bestaande abonnees. In deze lijst moet per abonnee worden vermeld: naam, adres, woonplaats en betaalwijze (week-, maand-, kwartaal- of jaarbetaling).
 2. periodiek moet een lijst worden afgedrukt van alle nieuwe abonnees en een lijst van alle abonnees waarvoor de betaalwijze is gewijzigd.
- Bij een analyse is o. a. gebleken dat wekelijks relatief veel mutaties optreden. Deze mutaties kunnen betrekking hebben op
- nieuwe abonnementen
 - adres- en/of woonplaatswijzigingen

- veranderingen in betaalwijze
- opzeggingen.

Op grond van de doelstellingen en de verrichte analyse is besloten:

- de relevante gegevens van de abonnees op te slaan in een sequentieel georganiseerd bestand, met abonneenummer als sleutelitem;
- de betaalwijze via het abonneenummer herkenbaar te doen zijn;
- het abonneebestand via een wekelijkse verwerking te onderhouden;
- de twee genoemde lijsten sub 2 elke acht weken te produceren.

- a. Is de sequentiële organisatie een terechte keuze voor het abonneebestand?
- b. Maak een ontwerp voor het abonneenummer zodat aan bovenstaande eisen (zo eenvoudig mogelijk) wordt voldaan.
- c. Ontwerp een recordindeling voor het mutatiebestand in verband met de wekelijkse verwerking.
- d. Maak een programmaschets voor de wekelijkse verwerking.

- 3.9. Maak een programmaschets voor het samenvoegen van drie sequentiële bestanden. (Zie opg. 3.4 voor een detailbeschrijving van een analoog geval van samenvoeging van twee bestanden.)

- 3.10. Gegeven zijn een orderbestand en een klantenbestand.

Elk orderrecord bevat o. a. ordernummer, klantnummer, repeating group bestelling (artikelcode, prijs, aantal). Het orderbestand is in volgorde van ordernummer sequentieel opgeslagen op magneetband.

Elk klantrecord bevat o. a. klantnummer, NAW, gegevens voor kredietwaardigheid-onderzoek. Het klantenbestand is in volgorde van klantnummer sequentieel opgeslagen op magneetband.

In de bewerking ORDER-KLANT dient elk order-record, mits voldaan is aan de kredietwaardigheid, te worden uitgebreid met NAW van de klant. Indien niet voldaan is aan de kredietwaardigheid, moet het desbetreffende order-record worden afgevoerd naar een lijst van niet te behandelen orders.

In voorgaande bewerkingen, waarbij overigens geen gebruik werd gemaakt van het klantenbestand, werden reeds alle mogelijke controles op fouten in het orderbestand uitgevoerd.

- a. Geef (in grote lijnen) weer hoe de bewerking ORDER-KLANT (zo efficiënt mogelijk) kan geschieden.
- b. Moet bij deze bewerking na alle voorgaande controles nog rekening worden gehouden met een foutenlijst?

- 3.11. Gegeven zijn twee orderbestanden OB_1 en OB_2 . Elk order-record (REC_1 resp. REC_2) van OB_1 resp. OB_2 bevat o. a. ordernummer ($ORNR_1$ resp. $ORNR_2$), klantnummer ($KLNR_1$ resp. $KLNR_2$), bedrag ($BEDR_1$ resp. $BEDR_2$). Eenzelfde klantnummer kan in beide bestanden voorkomen en/of in elke bestand meermalen voorkomen. Elk orderbestand is in oplopende volgorde van klantnummer op een magneetband opgeslagen. Met behulp van OB_1 en OB_2 moet een derde bestand B_3 worden gecreëerd. Elk record REC_3 van dit bestand bevat de volgende velden: klantnummer ($KLNR_3$), aantal orders (AANTAL), totaal bedrag (TOT-BEDR). In B_3 mag eenzelfde klantnummer maar eenmaal voorkomen. Per klantnummer dient AANTAL het totaal aantal orderrecords van de desbetreffende klant en TOT-BEDR het daarbij behorende totale bedrag te bevatten. B_3 moet sequentieel op klantnummer worden opgebouwd. Gevraagd een programmaschets voor de creatie van B_3 . Hierbij mag worden verondersteld dat bij een leesopdracht op OB_1 resp. OB_2 in de end of file toestand aan $KLNR_1$ resp. $KLNR_2$ de waarde ∞ wordt toegekend.
- 3.12. Gegeven zijn drie sequentieel georganiseerde bestanden B_1 , B_2 en B_3 . De eerste twee bestanden hebben dezelfde recordindeling (met sleutels S_1 resp. S_2). B_1 en B_2 hebben geen records gemeen, m. a. w. S_1 en S_2 hebben geen waarde gemeen. Sleutel S_3 (behorend bij de records van B_3) kan wel waarden gemeen hebben met S_1 of S_2 . Gevraagd wordt een programmaschets voor de opbouw van een sequentieel georganiseerd bestand B_4 dat een samenvoeging is van B_1 en B_2 met dien verstande dat records uit B_1 en B_2 , waarvan de sleutelwaarde ook in B_3 voorkomt, niet in B_4 worden opgenomen.
- 3.13. Gegeven is een stambestand STAM. Dit bestand is sequentieel opgeslagen op magneetband in oplopende sleutelvolgorde. Verder zijn gegeven de twee mutatiebestanden IN en UP. Het bestand IN bevat uitsluitend records die in STAM zullen worden ingevoegd. Het bestand UP bevat uitsluitend mutaties die uitgevoerd zullen worden op reeds bestaande records in STAM of op records die in IN voorkomen. Meerdere records met dezelfde sleutelwaarde kunnen wel voorkomen in UP, maar niet in IN. IN en UP zijn sequentieel opgeslagen op magneetband; de sleutelvolgorde is opklimmend. Gevraagd wordt een programmaschets voor de verwerking van de mutatiebestanden IN en UP. Hierbij dient rekening te worden gehouden met de volgende eventueel optredende fouten:
- Een sleutelwaarde in IN komt ook voor in STAM.
 - Een sleutelwaarde in UP komt niet voor in STAM en ook niet in IN.

Van goede mutaties moet een journaal-record (op band) worden geproduceerd. Foute mutaties moeten worden opgenomen in een foutenlijst (eveneens op band).

Bij het maken van de programmaschets is het niet toegestaan

- in een voorbereiding eerst IN en UP samen te voegen;
- het STAM-bestand meer dan eens te doorlopen.

- 3.14. Gegeven zijn twee klantenbestanden met dezelfde recordindeling. Het ene bestand A is een zogeheten actief bestand (records worden veelvuldig geraadpleegd), het andere bestand S is een zogeheten slapend bestand (records worden vrijwel niet geraadpleegd).

In elk klantenrecord komen o. a. voor het veld ltd (laatste transactiedatum), het veld saldo en het sleutelveld klnr (klantnummer). Voor elk record van bestand A geldt: $ltd \geq 780101$ òf $saldo \neq 0$; voor elke record van bestand S geldt: $ltd < 780101$ en $saldo = 0$.

Beide bestanden zijn sequentieel (op oplopend klnr) opgeslagen op magneetband. De bestanden A en S bevatten geen records met hetzelfde klnr.

- a. Verder is gegeven een mutatiebestand M, dat enkel veranderingen (updates) bevat (dus geen invoeringen en ook geen verwijderingen). Deze mutaties zijn bedoeld voor bestand A. Meer dan één mutatierecord is mogelijk bij eenzelfde klnr. Het bestand M is ook sequentieel (op oplopend klnr) opgeslagen.

Geef een programmaschets voor de verwerking van mutatiebestand M. Hierbij dient met het volgende rekening te worden gehouden:

- Als na verwerking van de mutaties op een bepaald record geldt: $ltd < 780101$ en $saldo = 0$, dan moet dit record uit bestand A worden verwijderd en in bestand S worden opgenomen.
- Als (enig mogelijke) fout kan optreden dat een klnr uit bestand M niet voorkomt in bestand A.
- Van een correcte resp. foute mutatie moet een journaal-record resp. foutmelding worden weggeschreven.

- b. (Dit gedeelte staat geheel los van het opgavegedeelte onder a.) Verder is gegeven een mutatiebestand M. Een mutatie kan bedoeld zijn als

- een invoeging van een nieuw klantenrecord in bestand A.
- een verandering op een bestaand record van bestand A of van bestand S.
- een verwijdering van een bestaand record in bestand S.

In elk mutatierecord is een veld mc (mutatiecode) dat aangeeft welke mutatie bedoeld is: in (insertion), up (update) of del (delete). Als mc = up is daaruit niet af te leiden of deze update bedoeld is op A of op S.

Het bestand M is sequentieel (op oplopend klnr en daarbinnen in de volgorde in, up, del) opgeslagen. Bij eenzelfde klnr kunnen uitsluitend de volgende combinaties van mutatierecords voorkomen:

- een invoeg-record gevolgd door n verandering-records ($n \geq 0$) òf
- n verandering-records ($n \geq 0$) òf
- een verwijdering-record, voorafgegaan door n verandering-records ($n \geq 0$).

Geef een programmaschets voor de verwerking van mutatiebestand M op de bestanden A en S. Hierbij dient rekening te worden gehouden met het volgende

- als na verwerking van de mutaties op een bepaald record van bestand A geldt: $ltd < 780101$ èn saldo = 0, dan moet record van bestand A naar bestand S worden overgeheveld. Omgekeerd moet een record van S naar A worden overgeheveld als ($ltd < 780101$ èn saldo = 0) niet meer geldt.
- als (enig mogelijke) fouten kunnen optreden:
 - . de sleutelwaarde van een in te voegen record komt reeds voor in bestand A of in bestand S.
 - . de sleutelwaarde van een veranderings-mutatierecord komt noch in bestand A, noch in bestand S voor.
 - . de sleutelwaarde van een te verwijderen record komt niet in bestand S voor.
- van een correcte resp. foute mutatie moet een journaal-record resp. foutmelding worden weggeschreven.

4 LIJSTSTRUKTUREN

4.1 INLEIDING

Om eventueel misverstand te voorkomen moeten wij bij het begin van dit hoofdstuk opmerken, dat een lijststructuur als regel niet bedoeld is als opslagstructuur voor een geheel bestand, maar voor een (klein) gedeelte van het bestand.

Lijststructuren (list structures) zijn in het algemeen gekarakteriseerd door het feit dat

- de plaats van een record in principe niet afhangt van de plaats van andere records;
- elk record via een (aaneenschakeling van) (adres)verwijzingen vanuit het eerste record bereikbaar is.

Uit het voorgaande volgt dat een lijststructuur een adresseerbaar geheugenmedium vereist.

Ter inleiding een eenvoudig voorbeeld.

Gegeven is een verzameling records met respectieve sleutelwaarden 3, 8, 12, 15 en 37. Deze records liggen opgeslagen op de respectieve geheugenadressen: 22, 15, 9, 36, 10 (zie figuur 4.1). Van elk record

(1)				
(5)				
(9)	12	37		
(13)			8	
(17)				
(21)		3		
(25)				
(29)				
(33)				15

Figuur 4.1. Verzameling records in geheugenruimte. Getallen tussen haakjes geven adressen weer. Getallen in de 'vakjes' geven sleutelwaarden weer.

is alleen de sleutelwaarde weergegeven. De inhoud van de geheugenvakken in de figuur, waarin niets vermeld staat, is irrelevant voor dit voorbeeld.

Als men de gegeven records in sleutelvolgorde zou willen benaderen, zou een adresverwijzing vanuit elk record naar zijn 'opvolger' op zijn minst erg nuttig zijn. De records zouden dan de volgende adresverwijzingen moeten bevatten.

record(sleutelwaarde)	adresverwijzing
3	15
8	9
12	36
15	10
37	-1

(Zie ook figuur 4.2.)

(1)				
(5)				
(9)	12 ₁	36 ₁	37 ₁	-1 ₁
(13)			8 ₁	9 ₁
(17)				
(21)		3 ₁	15 ₁	
(25)				
(29)				
(33)				15 ₁
				10 ₁

Figuur 4.2. Records met velden voor adresverwijzing.

Bij figuur 4.2 en ook volgende figuren wordt er vanuit gegaan dat in elk geheugenvak niet alleen ruimte is voor de eigenlijke recordvelden maar ook voor één of meer velden met adresverwijzingen.

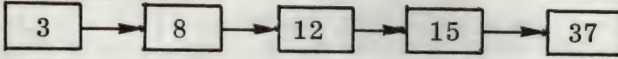
Naar het eerste record (met sleutelwaarde 3) bestaat in bovenstaand lijstje geen adresverwijzing en het laatste record (met sleutelwaarde 37) bevat geen (eigenlijke) adresverwijzing. Voor dit laatste zullen wij steeds de waarde -1 gebruiken. Waar nodig zal het adres van het eerste record op een aparte wijze worden bepaald (zie paragraaf 4.4).

Om adresverwijzingen tussen records in een tekening weer te geven, zullen wij de volgende afspraken maken:

- elk record wordt weergegeven in een zogeheten knoop;
- de verwijzing van een record naar een ander record wordt weergegeven met een zogeheten (gerichte) tak (pijl).

Bovengenoemde sequentiële volgorde wordt dus weergegeven in

de volgende tekening:

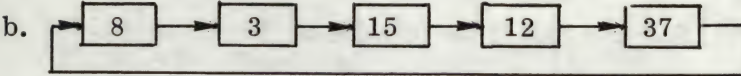


De plaats van een record in zo'n tekening zegt dus niets over zijn adres (de plaats in de geheugenruimte).

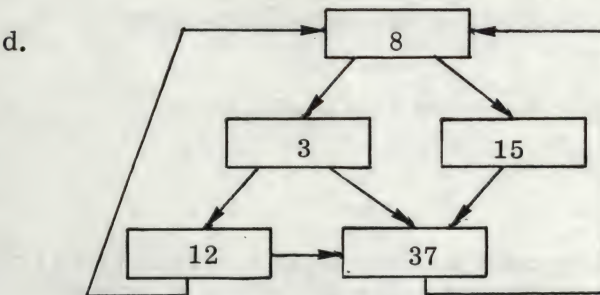
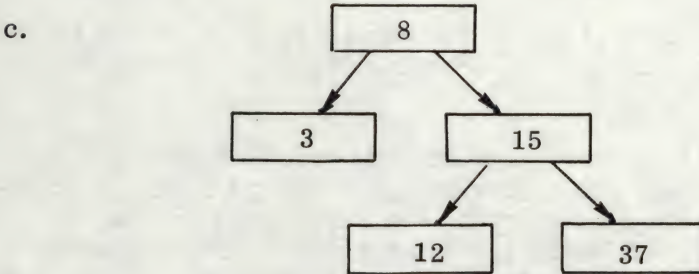
Tussen gegeven records zijn natuurlijk meer adresverwijzingen aan te brengen. Hieronder volgen enkele voorbeelden.



(bijv. opslag in volgorde van aanbidding)



(willekeurige volgorde en verbinding van 'laatste' met 'eerste' record)



Figuur 4. 3. Enkele voorbeelden van adresverwijzingen tussen records.

In figuur 4.3 c en d kan per record meer dan een adresverwijzing voorkomen en in figuur 4.3d kan naar één record via meer dan één adresverwijzing worden verwezen. Van voorbeeld d (vijf knooppunten en acht takken) wordt hieronder ook een 'beeld' van een mogelijke geheugenopslag gegeven.

(1)				
(5)				
(9)	12' 15' 10	37' -1' 15		
(13)			8' 22' 36	
(17)				
(21)		3' 9' 10		
(25)				
(29)				
(33)				15' 10' -1

Figuur 4.4. Records met meer dan een veld voor adresverwijzing per record (behorend bij figuur 4.3d).

4.2 SOORTEN LIJSTSTRUCTUREN

Een lijststructuur (list structure) is een structuur die met behulp van adresverwijzingen wordt vastgelegd.

Alvorens verder in te gaan op verschillende soorten lijststructuren, zullen wij eerst de begrippen voorganger en opvolger vastleggen met behulp van knopen en takken. Een voorganger van een knoop is een knoop, van waaruit een tak naar de desbetreffende knoop wijst; een opvolger van een knoop is een knoop, waarnaar een tak vanuit het desbetreffende knooppunt wijst. Zo is in figuur 4.3d knooppunt 37 een opvolger van 3 zowel als 15 en zijn 3 en 15 voorgangers van 37.

- Lijststructuren zijn te onderscheiden in drie hoofdsoorten:
- lineaire structuren
 - bomen
 - netwerken.

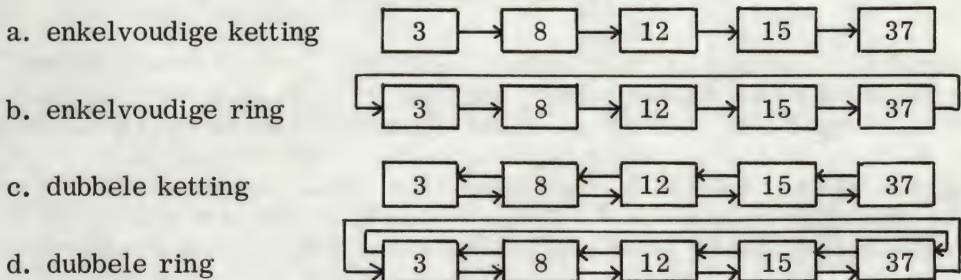
Lineaire structuren worden onderverdeeld in enkelvoudige en dubbele structuren. Bij beiden kent men dan nog de onderverdeling in ketting- en ringstructuur. (Voor lineaire structuren zullen wij soms de afkorting L. S. gebruiken.)

Een enkelvoudige L. S. is een structuur, waarin elk record

precies één voorganger en precies één opvolger heeft. In een ketting heeft het eerste record precies één opvolger en het laatste record precies één voorganger (zie figuur 4.5 a en b).

Een dubbele L. S. is een structuur, waarin elk record precies twee voorgangers en precies twee opvolgers heeft, met dien verstande dat de voorganger van een record ook opvolger is van datzelfde record en omgekeerd. In een ketting hebben het eerste en laatste record ieder precies één voorganger en precies één opvolger, met dien verstande dat de voorganger van een record gelijk is de opvolger van datzelfde record (zie figuur 4.5 c en d).

Het bovenstaande is in figuur 4.5 in beeld gebracht.



Figuur 4.5. Voorbeelden van lijststructuren.

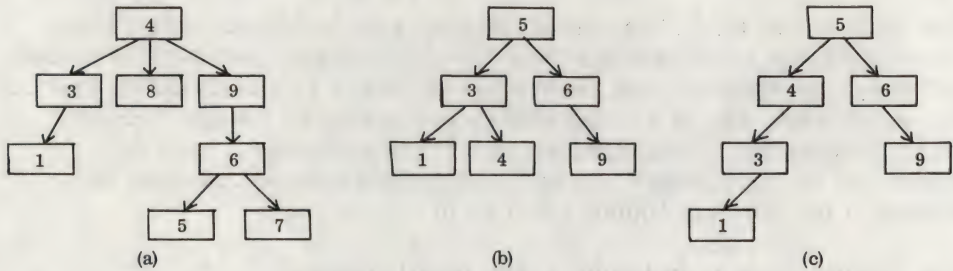
Er zij nogmaals op gewezen dat in figuur 4.5 met de tekening geen enkele informatie wordt gegeven over de plaats, waar de records liggen opgeslagen. Het 'naast elkaar liggen' in de figuur duidt enkel op adresverwijzing van een record naar een (op een willekeurig andere plaats gelegen) record.

In een lineaire structuur kunnen de records (in oplopende sleutelvolgorde) geordend of (in een willekeurige volgorde) ongeordend zijn. Een geordende structuur kennen wij ook onder de naam sequentiële structuur. Een ongeordende structuur wordt ook wel een seriële structuur genoemd.

Een variant op de ketting- en ringstructuur is die, waarbij de adresverwijzingen niet in de records zijn opgenomen, maar in een aparte wijzertabel (pointer-array). Ten aanzien van figuur 4.2 zou dit betekenen dat de adressen 15, 9, 36, 10, -1 (eventueel samen met de sleutels) als separate tabel worden opgeslagen. Aan deze variant zullen wij in dit boek verder niet apart aandacht besteden.

Een boom (tree) is een structuur waarin
- de beginknoop (top, wortel, root) geen voorganger heeft,

- elke knoop, behalve de beginknoop, precies één voorganger heeft,
 - elke knoop 0 of meer opvolgers heeft.
- (Zie figuur 4.3c en 4.6.)



Figuur 4.6. Bomen.

Bijzondere bomen zijn de zogeheten binaire bomen, waarin iedere knoop ten hoogste twee opvolgers heeft (4.3c, 4.6b en 4.6c). Het gebruik van een binaire boom zal in het algemeen efficiënter zijn naarmate de boom evenwichtiger is van opbouw. In dit verband is het begrip *balanced tree* van belang. Een binaire boom heet 'balanced' als voor elke knoop geldt dat het aantal knopen in de linkerboom hoogstens 1 verschilt met het aantal knopen in de rechter-subboom. Zo is de boom in 4.6b balanced, maar die in 4.6c niet.

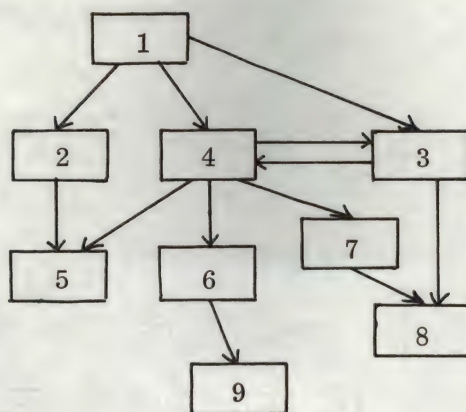
geval 4.6b

knoop	aantal knopen links	aantal knopen rechts
5	3	2
3	1	1
6	0	1
1	0	0
4	0	0
9	0	0

geval 4.6c

5	3	2
4	2	0
3	1	0
1	0	0
6	0	1
9	0	0

Een *netwerk* is een structuur, waarin elke knoop 0 of meer voorgangers en 0 of meer opvolgers kan hebben (zie figuur 4.3d en figuur 4.7).



Figuur 4.7. Netwerk.

Voorbeelden van netwerkstructuren zijn 'familiebomen' (waarbij steeds beide ouders opgenomen worden in plaats van de vader alleen, zoals in stambomen), planningsstructuren, samenstellingstekeningen van apparaten, explosielijsten, enz.

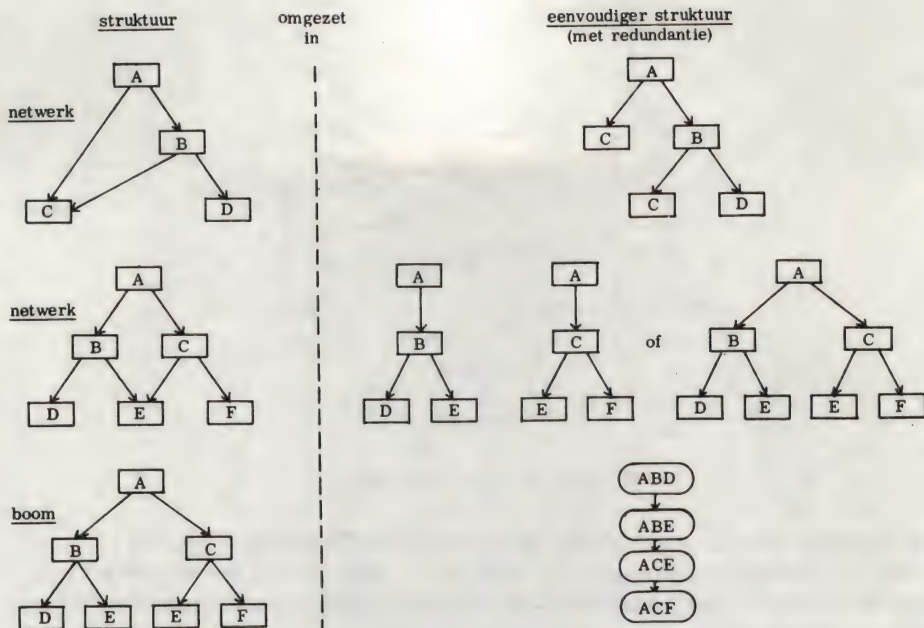
In een lijststructuur is sprake van een pad (path) van knoop a naar knoop b als men

- via één of meer takken van a naar b kan komen, en
- daarbij elke combinatie van naast elkaar liggende knopen niet meer dan éénmaal passeert.

Het aantal te doorlopen takken noemen we de lengte van het pad. Het pad zelf kan worden aangegeven met de achtereenvolgens te passeren knopen. Voorbeelden:

- In fig. 4.5a is er een pad van 8 naar 15 en wel (8, 12, 15), met een lengte van 2.
- In fig. 4.3a is geen pad van 15 naar 8.
- In fig. 4.5b is wel een pad van 15 naar 8, nl. (15, 37, 3, 8), dus met een lengte van 3.
- In fig. 4.5d zijn twee paden van 15 naar 8, met een lengte van 2 resp. 3.
- In fig. 4.7 bestaan van 1 naar 4 de paden (1, 4) en (1, 3, 4), maar (1, 4, 3, 4) is geen pad.
- In fig. 4.6b is de gemiddelde lengte per pad van de top naar elk der andere knooppunten 1, 6; in fig. 4.6c is deze gemiddelde lengte 1, 8.

Aan de hand van enkele figuren zullen we laten zien, dat door de invoering van overtoolligheid (redundancy) ingewikkelder structuren te vervangen zijn door eenvoudiger structuren (niet dat deze vervanging altijd na te streven zou zijn (!); naast overtoolligheid krijgen we problemen met de toegangstijd, gegevensbescherming, bijwerken van alle kopieën, enz.).

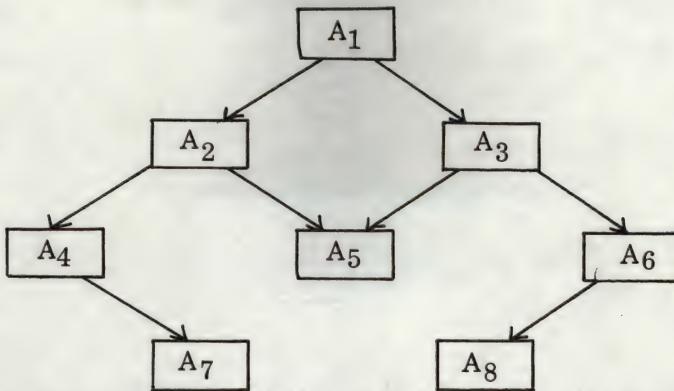


Opmerking:

Het is bij aan te schaffen apparatuur en programmatuur zaak te onderzoeken welke structuren mogelijk zijn, daar men anders zijn problemen op maat moet snijden door overtoolligheid in te voeren! Zo moest men in de tijd dat alleen magneetbanden als achtergrondgeheugens beschikbaar waren (dus alleen consecutieve opslagstructuur mogelijk was) boom- en netwerkstructuren 'platdrukken' tot lineaire structuren (zie laatste structuur in het voorbeeld).

Als toelichting op de vorige tekening en bovenstaande opmerking moge het volgende voorbeeld dienen. Met name zal hierbij het 'platdrukken' worden behandeld. We beschouwen daartoe het volgende 'netwerk' van activiteiten A_1 tot en met A_8 , die nodig zijn om een bepaald projekt tot stand te brengen. Voor degenen die bekend zijn met de 'netwerkplanning' zij hierbij opgemerkt dat in dit geval de knopen geen mijlpalen maar activiteiten voorstellen.

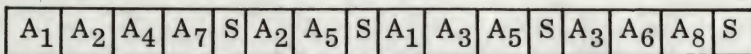
De logische betekenis van de pijlen is: activiteit aan de voet van de pijl moet beëindigd zijn voordat aan de activiteit aan de top van de pijl kan worden begonnen. Dus A_2 kan pas starten als A_1 is voltooid; A_5 kan pas starten na voltooiing van A_2 en A_3 . Elke activiteit A_i is op te vatten als een uitgebreid 'record' met allerlei informatie over de desbetreffende activiteit, zoals identificatie (nummer)/omschrijving/geplande duur/afgewerkt gedeelte/nog te verrichten gedeelte/in



te zetten werknemers/benodigde materialen/, etc.

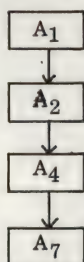
Hoe kan nu dit netwerk op een fysiek geheugen worden opgeslagen, zodat niet alleen de informatie per activiteit behouden blijft, maar ook de netwerkstructuur tussen de activiteiten? Als men een direct adresseerbaar geheugen heeft, kan men de records op evenzovele geheugenplaatsen ter grootte van een record opslaan. Voor weergave van de netwerkstructuur kan bijv. gebruik worden gemaakt van verwijzadressen in de records. In elk record moet dan voor elke uitgaande pijl een verwijzing staan naar het record op de top van die pijl. Dus in record A₂ moeten verwijzingen staan naar de adressen van record A₄ en record A₅.

Hoe moet men echter te werk gaan bij een niet-adresseerbaar geheugen als een magneetband? Stel dat de records achter elkaar (in welke volgorde dan ook) op band worden opgeslagen, dan is men de netwerkstructuur en dus de noodzakelijk aan te houden volgorde der activiteiten kwijt. Er zit hier niets anders op dan met redundantie te werken, dus records meer dan eenmaal op te slaan. Daartoe gaan we na welke paden, uitgaande van A₁ door het netwerk lopen. Deze paden zijn A₁ A₂ A₄ A₇; A₂ A₅; A₁ A₃ A₅; A₃ A₆ A₈. Als we nu deze paden opslaan op magneetband, dan is het netwerk dus 'platgedrukt' weergegeven door nl. elk pad 'plat te drukken'. Zie schets hieronder, waarin met S een scheiding tussen twee opeenvolgende paden wordt aangegeven.

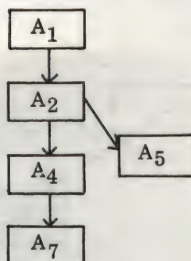


We hebben hiermee de netwerkstructuur 'reconstrueerbaar' gehouden, doch uiteraard ten koste van een grote mate van redundantie (12 recordplaatsen voor 8 verschillende records). Wil men nu het netwerk reconstrueren vanaf de op band vastgelegde informatie, dan leest men telkens een pad van de band en zet het netwerk weer 'pad-gewijze' op. Dus:

na lezing 1e pad:

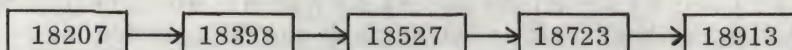


na lezing 2e pad:



4.3 COMBINATIE VAN CONSECUTIEVE STRUKTUUR EN LIJSTSTRUCTUREN

In voorkomende gevallen kan het zinvol zijn voor de opslag van een bestand een combinatie te kiezen van consecutieve en lijststructuur. Een voorbeeld hiervan is de invoeging van nieuwe records in een oorspronkelijk sequentieel georganiseerd bestand op een adresseerbaar medium, waarbij niet het gehele bestand wordt herschreven. In paragraaf 3.3 werd reeds gewag gemaakt van deze aanpak. Wij zullen het nu meer in detail beschrijven. Ieder in te voegen record wordt (aan het einde van het bestand) op een vrije plaats opgeslagen en in een enkelvoudige sequentiële ketting opgenomen die de records met elkaar verbindt. Zo'n ketting bevat alleen de nieuwe records, waarvan de sleutelwaarden zijn gelegen tussen twee opeenvolgende sleutelwaarden van het oorspronkelijke bestand. Stel dat in het oorspronkelijke bestand het record met sleutelwaarde 18207 wordt gevolgd door het record met sleutelwaarde 18913, dan zullen de in de loop der tijd in te voegen records met sleutelwaarden 18527, 18398, 18723 op beschikbare vrije plaatsen worden opgeslagen en met de records 18207 en 18913 in de volgende ketting worden opgenomen:



Deze strategie heeft onder meer tot gevolg, dat elk record in het oorspronkelijke bestand moet worden voorzien van een apart verwijzingsveld.

4.4 VOORBEELDEN VAN VERWERKING OP BASIS VAN EEN GEGEVEN LIJSTSTRUCTUUR

Als voor een verzameling records een structuur is vastgelegd, is het uiteraard zaak van deze structuur op de juiste wijze gebruik te maken en deze bij veranderingen in de verzameling records intact te laten.

We zullen dit toelichten aan de hand van twee voorbeelden. Het eerste voorbeeld betreft het gebruik, het tweede de verandering van een verzameling records. Bij beide voorbeelden veronderstellen we dat de verzameling records (als onderdeel van een bestand) ligt opgeslagen op een direct adresseerbaar achtergrondgeheugen. Om een record van het achtergrondgeheugen in het werkgeheugen in te lezen zal een leesopdracht nodig zijn. Hiervoor zullen wij gebruik maken van de leesopdracht voor niet-sequentiële verwerking (appendix II. 4). Voor het wegschrijven van een record zullen wij op analoge wijze gebruik maken van de schrijfoopdracht voor niet-sequentiële verwerking (appendix II. 4).

Wij kunnen nu beginnen met

VOORBEELD 1

Gevraagd een programmaschets voor het zoeken in een ongeordende enkelvoudige ketting en afdrukken van een record met een gegeven sleutelwaarde.

Analyse: Op grond van de gegeven sleutelwaarde zullen wij net zo lang 'zoeken' in de ketting, totdat het

- niet meer nodig is verder te zoeken aangezien het gevraagde record is gevonden
- en/of niet meer mogelijk is verder te zoeken, aangezien wij bij het zoeken het einde van de ketting (adresverwijzing = -1) hebben bereikt.

'Zoeken' betekent hier: een record inlezen en de sleutelwaarde hiervan vergelijken met de gegeven sleutelwaarde van het gezochte record. Heeft het zoekproces succes, dan moet het gevraagde record worden afgedrukt. Heeft het zoekproces geen succes, dan zal een tekst zoals 'niet gevonden' worden afgedrukt.

Met bovenstaande analyse ligt de opzet van de programmaschets wel vast. Wij zullen alleen nog enkele afspraken (voor dit en volgende problemen) moeten maken ten aanzien van

- bepaling van het adres van het eerste record in de ketting. Wij zullen hiervoor de procedure eerste(adres) gebruiken. Deze procedure levert het beginadres van de ketting. In het geval van een lege ketting zal deze procedure aan adres de waarde -1 toekennen. Wij laten de beschrijving van (de body van) deze procedure hier met opzet achterwege, omdat deze telkens verschillend kan zijn (zie met name hoofdstukken 7 en 8*).
- te gebruiken namen:
stbd: het stambestand; de ketting is een deel van dit bestand

*) Deze aanpak is verschillend van die in vorige drukken. Daar werd gewerkt met een zogeheten veld first. Deze aanpak lijkt ons minder goed aan te sluiten bij de volgende hoofdstukken.

- strec: stamrecord met de velden
 sl : sleutel
 data : (overige) gegevens
 verw: adresverwijzing
 zsl : zoeksleutel, een apart veld dat de sleutelwaarde bevat van
 het gevraagde record
- het inlezen van de sleutelwaarde van het gezochte record: hiervoor zullen wij de procedure accept(zsl) gebruiken
 - het afdrukken van het gevraagde record of de tekst 'niet gevonden'; hiervoor zullen wij de procedure print gebruiken.

Na bovenstaande voorbereidingen moge de bijgaande programmaschets verder duidelijk zijn.

```

begin type record strec(sl,data,verw);
      file stbd of strec
endtype;
var stb: stbd
endvar;

ready(stb);           %input, nodig voor zoekproces%

begin var sr: strec;
      zsl: sl; adres: verw; gevonden: boolean
endvar;
accept(zsl); gevonden := false;
eerste(adres);
while adres ≠ -1 and not gevonden
do read(stb,adres,sr);
  if sr.sl = zsl
  then gevonden := true
  fi;
  adres := sr.verw
od;
if gevonden
then print(sr)
else print("niet gevonden")
fi
end;
end

```


VOORBEELD 2

Gevraagd een programmaschets voor het invoegen van een (nieuw) record in een enkelvoudige sequentiële ketting.

Analyse: Wij zullen weer aannemen dat met de procedure eerste(adres) de variabele adres de waarde krijgt van het beginadres van de ketting met voor de lege ketting een waarde -1.

Er moet rekening worden gehouden met de volgende vijf invoeg-situaties:

A: invoegen in een lege ketting

B: invoegen in een niet-lege ketting

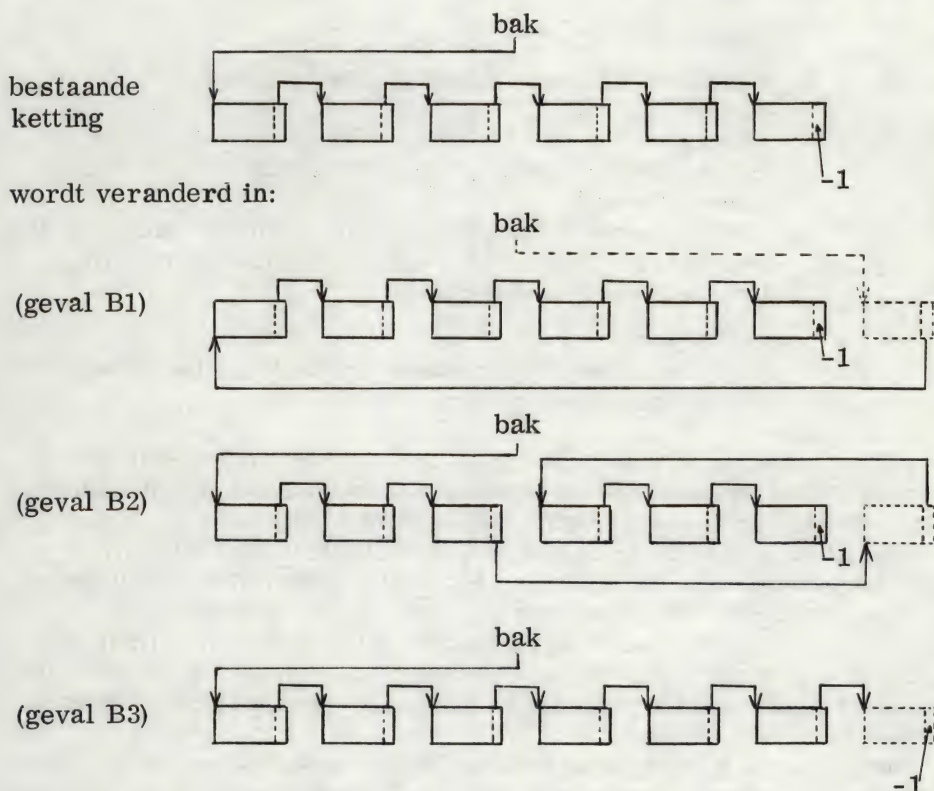
B1: als eerste record

B2: als 'tussen' record

B3: als laatste record

C: invoegen niet mogelijk wegens een reeds bestaande sleutelwaarde in de ketting

Zie voor de gevallen B1, B2 en B3 ook onderstaande schets.



bak: beginadres ketting; [dashed box]: ingevoegde record

Alvorens elk van bovengenoemde gevallen nader te ontleden zullen wij eerst nagaan hoe wij kunnen vastleggen welk geval in feite optreedt. Het ligt voor de hand hiervoor de ketting sequentieel af te lopen.

Wij beginnen dan met de bepaling van het beginadres van de ketting. Geeft dit de waarde -1 dan is de ketting leeg (geval A). Is de ketting niet leeg, dan lezen wij het eerste record om te kunnen vaststellen of het nieuwe record als eerste in de ketting moet worden opgenomen (geval B1). Is dit laatste niet het geval dan zullen we zolang het kan (dus nog niet het einde van de ketting is bereikt) en het ook nog zin heeft (dus de sleutelwaarde van het laatstgelezen record kleiner is dan die van het in te voegen record) het volgende record van de ketting lezen. Hierna zal sprake zijn van respectievelijk geval B2, B3, C als de sleutelwaarde van het laatstgelezen record respectievelijk groter dan, kleiner dan, gelijk aan die van het in te voegen record is. We kunnen de karakteriserende situaties voor elk geval kort samenvatten als in onderstaande tabel.

<u>geval</u>	<u>gekaracteriseerd door</u>
A	beginadres = -1
B	beginadres $\neq -1$ en
B1	sleutelwaarde eerste record groter dan sleutelwaarde nieuwe record
of B2	sleutelwaarde van voorlaatst- resp. laatstgelezen record kleiner resp. groter dan sleutelwaarde van nieuwe record
of B3	sleutelwaarde laatstgelezen record kleiner dan sleutel- waarde van nieuwe record en laatstgelezen record is laatste record in de ketting
C	beginadres $\neq -1$ en sleutelwaarde laatstgelezen record is gelijk aan sleu- telwaarde nieuwe record

Wij zullen aannemen dat voor het plaatsen van nieuwe records een voldoende aantal vrije adressen beschikbaar is voor het stambestand, waarin de ketting is opgenomen. Met de procedure bepaalvrij(stb, vrijadr) wordt aan vrijadr het adres van een vrije plaats toegekend. Als bij invoeging het beginadres van de ketting moet worden veranderd, dan gebruiken wij daarvoor de procedure nieuweerste(vrijadr). Als de vrije plaats, aangegeven in vrijadr, ook werkelijk wordt gebruikt voor het invoegen van een nieuw record, dan moet deze plaats als vrije plaats worden afgevoerd met de procedure verwijdervrij(stb, vrijadr).

De plaats van het nieuwe record is onafhankelijk van de 'logische plaats' in de ketting. Deze plaats wordt nl. aangewezen door vrijadr. De 'logische plaats' in de ketting zal wel van invloed zijn op de waar-

de van de adresverwijzing in het nieuwe record, en ook op het feit of en zo ja welke veranderingen in de bestaande ketting moeten plaatsvinden.

In onderstaand schema is voor elk van de genoemde mogelijkheden A t/m B3 weergegeven wat voorafgaande aan de invoeging,
 - in de bestaande schakels moet worden veranderd
 - de adresverwijzing moet zijn in het nieuwe record.

<u>geval</u>	<u>veranderingen in bestaande ketting</u>	<u>adresverwijzing in nieuwe record</u>
A	nieuw beginadres (vrijadr)	-1
B1	nieuw beginadres (vrijadr)	bestaand beginadres van ketting
B2	adresverwijzing in voorlaatst gelezen record veranderen in vrijadr (en wegschrijven!)	adresverwijzing uit voorlaatst gelezen record
B3	adresverwijzing in laatstgelezen record veranderen in vrijadr (en wegschrijven!)	-1

De invoeging, resp. foutmelding kan aan het einde van het programma plaatsvinden, als alle voorgaande akties (veranderingen in bestaande ketting en adresverwijzing in nieuwe record verzorgen) hebben plaats gevonden. Variabelen voor dit programma zullen zijn:

adres : adres in de ketting
 vadres : adres voorlaatst gelezen record
 hsr : hulpstamrecord, voor bewaren voorlaatst gelezen
 record
 inr : in te voegen record
 inv : invoeging, boolean die true is als invoeging mag door-
 gaan

```

begin type record strec(sl,data,verw);
      file stbd of strec
endtype;
var stb: stbd
endvar;

ready(stb);

begin var sr,hsr,inr: strec;
      adres,vadres,vrijadr: verw;
      inv: boolean
endvar;
accept(inr);           %in te voegen record inlezen%
bepaalvrij(stb,vrijadr);
inv := true;
eerste(adres);          %in adres begin ketting vastleggen%
if adres = -1           %geval A%
  then nieuweerste(vrijadr);          %nieuw beginadres%
      inr.verw := -1
  else read(stb,adres,sr);
      if sr.sl > inr.sl                %geval B1%
        then nieuweerste(vrijadr);    %nieuw beginadres%
            inr.verw := adres
        else while sr.sl < inr.sl and sr.verw ≠ -1
              do vadres := adres; hsr := sr;
                adres := sr.verw;
                read(stb,adres,sr)
              od;
            if sr.sl = inr.sl           %geval C%
              then inv := false
            else if sr.sl > inr.sl      %geval B2%
              then inr.verw := hsr.verw;
                  hsr.verw := vrijadr;
                  write(stb,vadres,hsr)
              else inr.verw := -1;      %geval B3%
                  sr.verw := vrijadr;
                  write(stb,adres,sr)
            fi
          fi
        fi;
      if inv
        then write(stb,vrijadr,inr); verwijdervrij(stb,vrijadr)
        else print("foutieve invoeging")
      fi
    end;
  save(stb)           %veranderde stb bewaren%
end

```


OPGAVEN

- 4.1. Geef van de gegevensstructuren in figuur 4.5 c en d een geheugenweergave, analoog aan figuur 4.4.

De volgende zes opgaven (4.2 tot en met 4.7) hebben alle betrekking op binaire bomen. De knopen worden geïdentificeerd met unieke (sleutel)-waarden. Voor de opbouw van de bomen geldt:

- vanuit elke knoop kunnen links uitgaande takken alleen verwijzen naar lagere sleutelwaarden en rechts uitgaande takken alleen naar hogere sleutelwaarden;
- toevoeging van een nieuwe knoop geschiedt steeds aan 'de uiteinden' van de boom.

Verder zullen we nog het begrip 'perfectly balanced' gebruiken. Een binaire boom is perfectly balanced als voor elke knoop geldt dat het aantal knopen 'ter linkerzijde' en 'ter rechterzijde' aan elkaar gelijk zijn.

- 4.2. Gegeven zijn vier elementen met waarden resp. 3, 5, 7 en 8. Zet deze elementen in een boom in de volgorde:
- a. 3. 5. 7. 8
 - b. 3. 7. 8. 5
 - c. 3. 7. 5. 8
 - d. 7. 3. 8. 5
 - e. 5. 8. 7. 3
- 4.3. Gegeven zeven elementen, genummerd 1 tot en met 7.
- a. Zet deze zeven elementen in een perfectly balanced tree.
 - b. Zet deze zeven elementen in een boom in de volgorde 4. 3. 5. 2. 1. 6. 7. Is deze boom balanced?
 - c. Wat is de gemiddelde lengte per pad van de top naar elk der andere knopen voor geval a en b?
- 4.4. Zet tien elementen, genummerd 1 tot en met 10, in een balanced tree.
- 4.5. Bij welk aantal elementen kan een boom perfectly balanced zijn?
- 4.6. a. Zet elk van de volgende rijen in de gegeven volgorde in een boom:
- a1. 10, 5, 8, 16, 20, 2, 13, 17, 14, 4, 1, 11, 9
 - a2. 2, 5, 4, 10, 8, 11, 14, 13, 9, 17, 20, 1, 16
- b. Wat is voor beide bomen de gemiddelde lengte per pad vanuit de top?
- 4.7. In welk opzicht zal de (gemiddelde) lengte der paden en dus 'de graad van balancerings' een rol spelen?

- 4.8. a. (Zie voorbeeld netwerkplanning in verband met 'platdrukken'.)
Voor een 'platte' weergave worden nu verder voorgesteld:

a1. $|A_1 A_2 A_4 A_5 A_7| S | A_1 A_3 A_5 A_6 A_8 |$

a2. $|A_1 A_2| S | A_1 A_3| S | A_2 A_4| S | A_2 A_5| S | A_3 A_5| S | A_3 A_6| S |$
 $|A_4 A_7| S | A_6 A_8| S |$

Wordt met a1 en a2 de netwerkstructuur 'behouden'? Zo ja, is een van deze vormen dan te verkiezen boven die van de eerdere tekst?

- b. Gegeven is dat een netwerkstructuur 'platgedrukt' als onderstaand wordt weergegeven volgens de beschreven methode:

$|A_1 A_2 A_8 A_9| S | A_1 A_3 A_6 A_9| S | A_1 A_4 A_5 A_6| S | A_1 A_4 A_5 A_7| S |$

Teken nu de bijbehorende structuur.

Ontwerp een programmaschets voor:

- 4.9. het invoegen van een record (aan het einde van) een ongeordende enkelvoudige ketting.
- 4.10. het zoeken van een record in een sequentiële enkelvoudige ketting.
- 4.11. het zoeken van een record in een dubbele sequentiële ketting.
- 4.12. het invoegen van een record in een dubbele sequentiële ketting.
- 4.13. het verwijderen van een record uit een ongeordende enkelvoudige ketting.
- 4.14. het verwijderen van een record uit een dubbele sequentiële ketting.
- 4.15. het zoeken van een record in een binaire boom (opgezet volgens opgaven 4.2 t/m 4.7).
- 4.16. het invoegen van een record in een binaire boom (zie 4.15).
- 4.17. Gegeven is een sequentieel georganiseerd bestand op magneetschijf. In te voegen records worden op vrije plaatsen opgeslagen. Ingevoegde records, waarvan de sleutelwaarden gelegen zijn tussen twee opeenvolgende sleutelwaarden van het oorspronkelijke bestand, worden in een sequentiële ketting met elkaar verbonden. Het record met de laagste van de twee genoemde sleutelwaarden in het oorspronkelijke bestand bevat het beginadres van de ketting (waarde -1 bij lege ketting). Het laatste record van de ketting heeft als adresverwijzing -1; wijst dus

niet terug naar het oorspronkelijke bestand. Verwijdering vindt als volgt plaats:

- van een record in het oorspronkelijke bestand, door een speciaal boolean veld weg de waarde true te geven (voor een bestaand record is deze waarde dus false);
 - van een record in een ketting, door het record uit die ketting te verwijderen.
- a. Geef een programmaschets voor verwerking van een mutatiebestand met alleen veranderingen (met veronderstellingen analoog aan A1-A6 in hoofdstuk 3).
- b. Geef een programmaschets voor verwerking van een mutatiebestand met alle drie mutatiesoorten (met veronderstellingen analoog aan B1-B6 in hoofdstuk 3).

5 SORTEREN EN ZOEKEN

5.1 INLEIDING

In hoofdstuk 3 is de sequentiële bestandsorganisatie behandeld. Daarbij is er steeds van uitgegaan, dat het stambestand en de mutatiebestanden op de juiste wijze zijn gesorteerd. Met name voor een mutatiebestand zal daartoe dikwijls een aparte sorteergang nodig zijn. In dit hoofdstuk zullen wij daarom, zij het beperkt, op sorteren ingaan. De opzet is namelijk alleen de lezer vertrouwd te maken met de principes van het sorteren en met enkele termen.

In hoofdstuk 3 werd ook gesteld dat de sequentiële bestandsorganisatie niet geschikt is als het gaat om het zoeken van één bepaald record op basis van een gegeven sleutelwaarde. In de nog te bespreken vormen van bestandsorganisatie zullen wij echter zien dat

- a. gedeelten van bestanden vaak sequentieel (of ongeordend) worden opgeslagen;
- b. hulpbestanden worden gebruikt, die zelf sequentieel worden georganiseerd;
- c. in de gevallen a en b het zoeken van één bepaald record op grond van een gegeven sleutelwaarde wel relevant is.

In het tweede gedeelte van dit hoofdstuk zal dan ook enige aandacht worden besteed aan het zoekprobleem.

5.2 SORTEREN

Onder het sorteren van records wordt verstaan het plaatsen van deze records in een voorgeschreven (logische) volgorde, meestal naar stijgende of dalende sleutelwaarde. Aan het sorteerproces zullen efficiëntie-eisen gesteld moeten worden, vooral als men bedenkt dat bij administratieve gegevensverwerking veel moet worden gesorteerd (volgens sommige schattingen 50% van de totale verwerkingstijd).

Wanneer het sorteren met de hand (bijv. met kaarten) gebeurt, worden de informatiedragers van de oorspronkelijke plaatsen in het ongesorteerde bestand op de uiteindelijke plaatsen in het gesorteerde bestand gezet (eventueel via een aantal 'tussenplaatsen'). Gebeurt dit

door middel van een rekenmachine dan wordt de informatie van de ene naar de andere plaats in een geheugen gekopieerd. Om in dit laatste geval een machine-onafhankelijke vergelijking van sorteerprocessen mogelijk te maken, wordt het gemiddelde aantal bewerkingen bepaald dat nodig is om uitgaande van een willekeurig geordend bestand het gesorteerde bestand te verkrijgen. Deze bewerkingen zullen bestaan uit het vergelijken van sleutelwaarden en het verplaatsen van (gedeelten van) records.

Sorteermethoden worden meestal gesplitst in twee klassen, de zg. 'interne' sorteermethoden, waarbij alle te sorteren records tegelijk in het werkgeheugen (kunnen) staan, en de 'externe' sorteermethoden, waarbij niet alle records tegelijk in het werkgeheugen kunnen worden opgenomen. In dit laatste geval zal dus tijdens het sorteerproces een gedeelte (meestal zelfs het grootste gedeelte) der records op een achtergrondgeheugen staan. Deze relatief langzame achtergrondgeheugens maken de meeste interne sorteermethoden onbruikbaar en vragen dus om een eigen aanpak.

We zullen nu eerst de interne sortering en daarna de externe sortering bespreken. Deze bespreking is zeker niet bedoeld als een diepgaande of volledige behandeling, waarvoor zij verwezen naar D. E. Knuth, 'The art of computer programming' vol. 3, met maar liefst 400 bladzijden besteed aan sorteren. Interessant is een citaat: "It would be nice if only one or two of the sorting methods would dominate all of the others, regardless of the application or the computer being used. But, in fact, each method has its own peculiar virtues. Thus we find that nearly all of the algorithms deserve to be remembered, since there are some applications in which they turn out to be best." (blz. 379). De laatste raadgeving valt buiten het bestek van dit boek.

5.2.1 Interne sortering

We zullen enkele eenvoudige technieken van interne sortering bespreken.

- Een invoegmethode
- Een selectiemethode
- Een samenvoegmethode (vanwege het belang voor externe sortering).

We zullen bij elk van de te behandelen methoden eenvoudigheidshalve aannemen dat

- de sleutels integers zijn
- de records slechts uit sleutels bestaan en opgeslagen zijn in een integer array $A[0:n]$, waarbij $A[0] = -1$, dus geen sleutelwaarde is
- steeds wordt gesorteerd naar opklimmende sleutelwaarden.

EEN INVOEGMETHODE

De rij te sorteren elementen wordt gesplitst in een reeds gesorteerd stuk, aanvankelijk uit het eerste element bestaand, en een nog te sorteren stuk. Het element in het ongesorteerde stuk, waar een wijzer naar wijst, wordt telkens op de goede plaats in het gesorteerde stuk tussengevoegd, waarbij in het algemeen een gedeelte van deze rij één plaats moet opschuiven. Met het oog op dit opschuiven wordt de wijzer geïnitieerd op het tweede element. Wij zijn klaar wanneer de hele rij geordend is, dus als de wijzer wijst voorbij het laatste element.

Voor het programma voor deze invoegmethode zullen wij ervan uitgaan dat het array A (globaal) gegeven is. Wij zullen hierbij de volgende variabelen nodig hebben:

w : wijzer, zoals boven beschreven

h : hulpvariabele, om een aan de beurt zijnde sleutelwaarde vast te houden

i : index van array-element, dat eventueel voor 'opschuiven' in aanmerking komt.

Hieronder volgt nu het programma:

```

begin var w,h,i: integer
      endvar;
      w := 2;
      while w ≤ n
        do h := A[w];           %in te voegen element%
          i := w-1;
          while h < A[i] and i ≥ 1
            do                   %element opschuiven%
              A[i+1] := A[i]; i := i-1
            od;
          A[i+1] := h;          %nieuwe element op juiste plaats%
          w := w+1
        od
      end

```

Opmerking. Met dit programma zal duidelijk zijn waarom een pseudo-element $A[0]$ aan het array is toegevoegd. Zonder $A[0]$ zou men anders in de binnenste lus bij $h < A[i]$ te maken kunnen krijgen met een ongedeclareerd element.

Voor n elementen hebben we n geheugenplaatsen nodig. De geheugenbezetting is dus evenredig met n , of anders gezegd van de orde n , $O(n)$.

Wanneer m elementen reeds gesorteerd zijn zal men gemiddeld $\frac{1}{2}m$ vergelijkingen moeten maken om de plaats te vinden waar het volgende element moet komen te staan. Vervolgens zijn er gemiddeld $\frac{1}{2}m$

opschuivingen nodig om deze plaats vrij te maken. Het totale aantal vergelijkingen en opschuivingen is dus evenredig met

$$\sum_{m=1}^{n-1} \frac{1}{2}m \approx n^2/4.$$

EEN SELECTIEMETHODE

Een andere voor de hand liggende sorteermethode berust op de gedachte een gesorteerde rij elementen op te bouwen door telkens het kleinste element van een ongesorteerd gedeelte van het array toe te voegen aan reeds eerder gevonden elementen van een gesorteerd gedeelte. Wil men het aantal gebruikte geheugenplaatsen echter $O(n)$ houden, dan moet een verwisseling plaatsvinden van het kleinste element van het ongesorteerde stuk met het element direct volgend op het gesorteerde stuk. Aanvankelijk is het gehele array ongesorteerd. Een programma voor deze methode is dan bijvoorbeeld:

```

begin var w,h,g,i: integer
      endvar;
      w := 0;
      while w < n-1          %laatste element hoeft niet onderzocht te worden%
      do w := w+1;           %w is plaats v.h. toe te voegen element%
        h := A[w]; g := w;
        i := w;
        while i < n          %kleinste element zoeken%
        do i := i+1;         %in ongesorteerd stuk%
          if A[i] < h then
            h := A[i]; g := i
          fi
        od;
        A[g] := A[w]; A[w] := h      %verwisseling%
      od
end

```

Vergelijken we deze selectiemethode met de invoegmethode, dan zien we dat het aantal verwisselingen weliswaar afgenomen is, maar het aantal vergelijkingen is nog steeds $O(n^2)$. Ook andere elementaire methoden waarbij in een 'sorteerslag' uiteindelijk hoogstens twee elementen verwisseld worden, blijken van de orde $O(n^2)$ te zijn, hetgeen voor grote n geen aangenaam vooruitzicht is. Voor een betere prestatie moet men het blijkbaar zoeken in methoden waarbij in een sorteerslag meer dan 2 elementen van plaats verwisselen. Zulke methoden, waarvan de zg. heapsort en quicksort voorbeelden zijn, blijken inderdaad beter te zijn, nl. van de orde $(n \log n)$. In het bestek van dit boek zou het echter te ver voeren deze methoden te bespreken. Wel zullen

we nu een interne sorteermethode bespreken die tevens de basis is voor externe sorteermethoden.

SAMENVOEGMETHODE (MERGING)

Bij de samenvoegmethode worden telkens een aantal gesorteerde deelrijen (aanvankelijk misschien uit 1 element bestaande) tot een nieuwe gesorteerde deelrij samengevoegd. Als samenvoeging plaatsvindt met telkens twee, drie, resp. p deelrijen, dan spreekt men van een 2-way merge, 3-way merge resp. p -way merge. Door opeenvolgende sorteerslagen krijgt men steeds langere gesorteerde deelrijen totdat tenslotte één gesorteerde rij ontstaat.

Men kan nog op twee verschillende manieren te werk gaan, nl. volgens de zogeheten natural merge of volgens de straight merge. Bij de natural merge maakt men gebruik van mogelijk reeds bestaande gesorteerde deelrijen, bij de straight merge worden de gesorteerde deelrijen volgens een vast stramien gemaakt.

Wij zullen het voorgaande toelichten met een aan Knuth ontleende (blz. 114 en 115) rij van 16 sleutelwaarden (gesorteerde deelrijen worden door horizontale streepjes afgesloten).

Gegeven rij	Natural merge				Straight merge			
	deel- rijen	eerste merge	tweede merge	derde merge	eerste merge	tweede merge	derde merge	vierde merge
503	<u>503</u>	087	061	061	087	061	061	061
087	<u>087</u>	503	087	087	<u>503</u>	087	087	087
512	<u>512</u>	<u>512</u>	170	154	<u>061</u>	503	170	154
061	<u>061</u>	061	503	170	<u>512</u>	<u>512</u>	275	170
908	<u>908</u>	170	512	275	<u>170</u>	<u>170</u>	503	275
170	<u>170</u>	897	897	426	<u>908</u>	275	512	426
897	<u>897</u>	<u>908</u>	<u>908</u>	503	<u>275</u>	897	897	503
275	<u>275</u>	<u>275</u>	154	509	<u>897</u>	<u>908</u>	<u>908</u>	509
653	<u>653</u>	426	275	512	<u>426</u>	<u>154</u>	<u>154</u>	512
426	<u>426</u>	<u>653</u>	426	612	<u>653</u>	426	426	612
154	<u>154</u>	<u>154</u>	509	653	<u>154</u>	509	509	653
509	<u>509</u>	509	612	677	<u>509</u>	<u>653</u>	612	677
612	<u>612</u>	612	653	703	<u>612</u>	<u>612</u>	653	703
677	<u>677</u>	677	677	765	<u>677</u>	677	677	765
765	<u>765</u>	703	703	897	<u>703</u>	703	703	897
703	<u>703</u>	<u>765</u>	<u>765</u>	<u>908</u>	<u>765</u>	<u>765</u>	<u>765</u>	<u>908</u>

Opmerkingen:

- bij de straight merge bestaan de oorspronkelijke deelrijen eigenlijk allemaal uit precies één element;
- het ligt voor de hand dat voor het (eventuele) voordeel van de natural merge een prijs moet worden betaald in de vorm van een meer ingewikkeld programma.

Het aantal bewerkingen (vergelijkingen en verplaatsingen) dat nodig is voor een straight p -way merge is gelijk aan $np \text{ entier}(P \log n + 1)$, waarbij n het aantal te sorteren elementen is. Dit is als volgt in te zien.

In de eerste merge slag zullen wij n/p gesorteerde rijtjes ter lengte p maken. In de tweede slag krijgen we rijtjes ter lengte p^2 , enz.; zodat het proces ophoudt in x slagen wanneer $p^x > n$. M.a.w. het aantal sorteerslagen is $P \log n$ (uiteraard naar boven afgerond tot een geheel getal). Het aantal vergelijkingen per slag is $n(p-1)$, aangezien elke nieuwe plaatsing wordt voorafgegaan door $p-1$ vergelijkingen. Het aantal vergelijkingen is dus $n(p-1) \text{ entier}(P \log n + 1)$. Het aantal verplaatsingen is n maal het aantal slagen. De totale sorteertijd is dus $np \text{ entier}(P \log n + 1)$.

5.2.2 Externe sortering

Wanneer een bestand zo groot is dat het niet in zijn geheel in het werkgeheugen kan worden opgenomen, heeft dit ingrijpende consequenties. Deze omstandigheid zal zich bij verreweg de meeste bestanden voordoen. We zullen dan moeten sorteren met behulp van externe geheugens en spreken dan ook van externe sortering. Aangezien de toegangsmogelijkheden bij externe geheugens nogal sterk verschillen van die bij interne geheugens, zullen we in het algemeen de interne sorteermethoden niet kunnen gebruiken. Van het uitgebreide gebied van externe sortering zullen we twee methoden bespreken en wel

- p -way merge sortering en
- polyphase merge sortering.

In beide gevallen nemen we magneetbanden als externe geheugens aan. Tenslotte zullen wij een externe sorteermethode bespreken voor een adresseerbaar achtergrondgeheugen.

P-WAY MERGE SORTING

Evenals bij interne merge sorting kunnen wij hier onderscheid maken tussen natural en straight merge sorting. Wij zullen slechts de natural two-way sortering nader bespreken. Hiervoor zijn 3 magneetbanden nodig, doch met 4 gaat het sneller. De oorspronkelijke gegevens worden van de (invoer)band gelezen en zolang de sleutelvolgorde klopt met de sorteervolgorde worden de records weggeschreven naar de eerste (uitvoer)band. Klopt de volgorde niet meer, dan wordt weg-

geschreven naar de tweede (uitvoer)band. Bij onjuiste volgorde wordt weer weggeschreven naar de eerste uitvoerband, enz. Dit proces wordt voortgezet totdat de invoerband in zijn geheel gelezen is. Nu wordt de invoerband meestal afgenomen en op de twee vrije magneetbanden worden nieuwe banden geplaatst. De twee uitvoerbanden worden nu invoerbanden en door middel van een two-way merge wordt een (gesorteerde) deelrij van invoerband 1 en invoerband 2 samengevoegd tot een nieuwe (gesorteerde) deelrij die op de (nu) eerste uitvoerband komt te staan. De volgende twee deelrijen worden op de (nu) tweede uitvoerband geplaatst, de hierna volgende weer op de eerste, enz. Zijn zo alle deelrijen samengevoegd, dan wisselen uit- en invoerband weer van functie, enz. Dit proces wordt voortgezet totdat het gehele bestand gesorteerd is. Beschikken we over magneetbanden die ook 'achteruit' gelezen kunnen worden (backward read), dan kunnen we bij bovengenoemde funktiewisseling de terugspoeltijd (rewind time) besparen.

Het aantal merge slagen kan verminderd worden door bij de aanvankelijke uitsplitsing eerst zoveel mogelijk records in het werkgeheugen in te lezen, deze te sorteren met behulp van een interne sorteertechniek, en de aldus gevormde deelrijen afwisselend op de eerste en de tweede uitvoerband weg te schrijven. Het aantal merge slagen dat nu nodig is kan gemakkelijk berekend worden met behulp van een soortgelijke redenering als bij de interne merge sort.

Indien $2p$ magneetbanden beschikbaar zijn gebeurt de uitsplitsing van het oorspronkelijke bestand over p magneetbanden. De deelrijen op deze p magneetbanden worden met een p -way merge samengevoegd. Iedere nieuwe samengevoegde rij wordt op de volgende magneetband weggeschreven (geteld wordt modulo p). Het gehele sorteerproces vereist nu minder merge slagen. De relatieve tijdwinst voor dit proces is in de praktijk toch echter minder dan de relatieve vermindering van het aantal merge slagen in vergelijking met de two-way merge. Dit komt omdat nu p input buffers in het werkgeheugen aanwezig moeten zijn in plaats van 2, met het gevolg dat de blokkingsfactor kleiner zal moeten zijn, en dit maakt de totale lees/schrijftijd per merge slag groter.

VOORBEELD

De records van het voorbeeld in 5.2.1 extern te sorteren met een

- two-way merge
- three-way merge.

Bij een two-way merge zijn de volgende stappen te onderscheiden:

I (vanaf invoerband)		II		III		IV
Bd3	Bd4	Bd1	Bd2	Bd3	Bd4	records gesorteerd op Bd1
<u>503</u>	087	087	061	061	154	
061	<u>512</u>	503	170	087	275	
<u>908</u>	170	<u>512</u>	897	170	426	
275	<u>897</u>	275	<u>908</u>	503	509	
<u>653</u>	<u>426</u>	426	154	512	612	
154	<u>703</u>	<u>653</u>	509	897	653	
509			612	<u>908</u>	677	
612			677		703	
677			703		<u>765</u>	
<u>765</u>			<u>765</u>			

Bij een three-way merge:

I (vanaf invoerband)			II			III
Bd4	Bd5	Bd6	Bd1	Bd2	Bd3	records gesorteerd op Bd4
<u>503</u>	087	061	061	170	154	
170	<u>512</u>	<u>908</u>	087	275	509	
<u>897</u>	275	<u>426</u>	503	426	612	
154	<u>653</u>		512	653	677	
509	<u>703</u>		<u>908</u>	<u>897</u>	703	
612					<u>765</u>	
677						
<u>765</u>						

POLYPHASE MERGE SORTERING

Een nadeel van de voorgaande methode is dat slechts de helft van het aantal bandeenheden deelneemt aan het merge proces, van de andere helft wordt telkens maar één eenheid gebruikt als uitvoermedium, de rest is tijdelijk inactief. Bij de polyphase sortering worden alle bandeenheden op één na gebruikt als invoermedium bij het merge proces. De ene overblijvende dient als uitvoerband.

VOORBEELD

Polyphase merge sortering met het voorbeeld van 5.2.1 met gebruik van 3 magneetbandeenheden (zie ook de tabel op p. 84). We moeten starten met 5 deelrijen op band 1 en 3 deelrijen op band 2 (kolom I). De laatste worden gemengd met de eerste drie deelrijen van band 1 en

het resultaat wordt opgeslagen op band 3 (kolom II, waar ook het restant van band 1 is aangegeven). Band 2 is nu beschikbaar om het resultaat van de mergeslag van banden 1 en 3 op te nemen. Samen met het restant van band 3 is dit in kolom III aangegeven. Het restant van band 3 wordt dan gemengd met de eerste deelrij van band 2; het resultaat komt op band 1; het restant van band 2 blijft daar. Tenslotte geeft merging van band 1 en band 2 het resultaat op band 3.

I		II		III		IV		V
Bd1	Bd2	Bd1	Bd3	Bd2	Bd3	Bd1	Bd2	records gesorteerd op Bd3
<u>503</u>	087		087	087		087		
061	<u>512</u>		503	154		154		
908	170		<u>512</u>	503		275		
275	<u>897</u>		061	509		426		
653	<u>426</u>		170	512		503		
154		154	897	612		509		
509		509	<u>908</u>	677		512		
612		612	275	<u>765</u>	275	612		
677		677	426	061	426	653	061	
765		<u>765</u>	<u>653</u>	170	<u>653</u>	677	170	
703		<u>703</u>		703		<u>765</u>	703	
				897			897	
				<u>908</u>			<u>908</u>	

Opmerkingen:

- Aan het geschetste voorbeeld zien we dat hier meer merge slagen nodig zijn dan voor een proces dat met 4 bandeenheden een two-way merge uitvoert. We moeten echter wel bedenken dat per merge slag niet alle records meedoen; hierdoor kan toch een verkorting van de sorteertijd optreden.
- Polyphase sorting is alleen mogelijk bij bepaalde initiële verdelingen van de deelrijen over de invoerbanden. In het behandelde voorbeeld vereist dit 5 deelrijen op band 1 en 3 deelrijen op band 2. Met 4 deelrijen op iedere invoerband loopt men bijvoorbeeld vast. Op de achtergrond van deze initiële verdeling gaan we verder niet in.

EXTERNE SORTERING OP ADRESSEERBARE ACHTERGROND-GEHEUGENS

Voor externe sortering op adresseerbare achtergrondgeheugens zullen wij ons hier beperken tot de bespreking van één methode, nl. die waarbij sortering van de sleutelwaarden met bijbehorende adressen plaatsvindt zonder daarbij tijdens het sorteerproces de rest van de gegevens in de records te betrekken. Dit kan, met name bij grote records, veel transporttijd besparen.

In deze methode gaat men nu als volgt te werk. Aanvankelijk wordt elk record ingelezen en wordt van elk record de sleutelwaarde en het adres in een tabel opgeslagen. Wij nemen voorlopig aan dat deze tabel in zijn geheel in het interne geheugen kan worden opgenomen (zie de opgaven voor het geval dit niet mogelijk is). Vervolgens wordt deze tabel, met een of andere interne sorteermethode, op sleutelwaarde gesorteerd. Met behulp van de tabel wordt tenslotte het bestand met de volledige records gesorteerd door de gesorteerde tabel met de paren (sleutelwaarde, adres) sequentieel af te werken. Per paar wordt dan het record op het bijbehorende adres ingelezen en weggeschreven naar het nieuwe (gesorteerde) bestand.

5.3 ZOEKMETHODEN

Bij het zoeken van een record met een gegeven sleutelwaarde kan het van belang zijn of de verzameling, waarin gezocht moet worden, geordend is of niet.

Ter toelichting het volgende voorbeeld. Stel dat de records met de sleutelwaarden 091, 007, 115, 130, 083, 070, 105 in deze volgorde, dus ongeordend, toegankelijk zijn. Wil men het record met sleutelwaarde 083 lezen, dan zullen eerst de records met de sleutelwaarden 091, 007, 115, 130 moeten worden geïnspecteerd. Zou worden gevraagd naar het record met sleutelwaarde 009, dan zal pas na inspectie van alle records kunnen worden geconcludeerd dat dit record niet aanwezig is. Zouden de records echter in oplopende volgorde van sleutelwaarde (dus in de volgorde 007, 070, 083, 091, 105, 115, 130) toegankelijk zijn, dan zou al na inspectie van het tweede record blijken dat het record met sleutelwaarde 009 niet aanwezig is.

Overigens is niet alleen het geordend zijn van belang voor het zoekproces maar ook of we te maken hebben met een consecutieve of een lijststructuur, en welk opslagmedium wordt gebruikt.

Wij zullen het zoekproces dan ook beschouwen, voor verschillende structuren en opslagmedia; met name komen ter sprake het zoeken van een record in een

- a. ongeordende structuur
- b. geordende consecutieve structuur
 - b.1. op een niet-adresseerbaar medium
 - b.2. op een adresseerbaar medium
- c. een geordende kettingstructuur.

Ad a. Het zoeken in een ongeordende structuur kan alleen met de zogeheten lineaire zoekmethode, dat wil zeggen, de records worden achtereenvolgens vanaf het begin doorlopen. Het maakt hierbij verder niet uit of sprake is van een consecutieve of een kettingstructuur. Als maat voor de 'lengte' van het zoekproces zal worden aangehouden het aantal malen dat de opgegeven sleutel (van het gezocht

te record) wordt vergeleken met de sleutel (van het laatst benaderde record). Bij een ongeordende structuur zullen voor het zoeken van een record gemiddeld $(n+1)/2$ vergelijkingen nodig zijn (n = totaal aantal records), wanneer het te zoeken record aanwezig is. Is het te zoeken record niet aanwezig dan zullen zonder meer n vergelijkingen nodig zijn. Immers pas na n vergelijkingen kan worden vastgesteld dat het gezochte record niet aanwezig is. Als de kans op niet aanwezig zijn van een record bijv. gelijk is aan p , dan is het te verwachten aantal vergelijkingen voor het zoeken van een willekeurig record gelijk aan:

$$(1-p) \frac{n+1}{2} + pn \approx \frac{1+p}{2} n$$

Ad b.1. (geordende consecutieve structuur op niet-adresseerbaar medium)

Ook hier kan het zoekproces alleen via de lineaire zoekmethode plaatsvinden. Aangezien het nu echter een lineaire zoek op een geordende verzameling records betreft, zal de 'lengte' gegeven worden door

$$\frac{n+1}{2} \approx \frac{n}{2}$$

Ad b.2. (geordende consecutieve structuur op adresseerbaar medium)

Uiteraard kan ook nu de lineaire zoekmethode worden toegepast. In dit geval is echter een andere (in het algemeen meer efficiënte methode) beschikbaar, en wel de zogeheten binaire zoekmethode. Om deze te kunnen toepassen moeten de records wel een vaste lengte hebben.

Bij de binaire zoekmethode gaat men als volgt te werk. Begonnen wordt met het (ongeveer) middelste record van de totale verzameling records. Nu kunnen zich drie gevallen voordoen:

$zsl < msl$	waarbij zsl : sleutel van gezochte record
$zsl = msl$	msl : sleutel van het middelste record.
$zsl > msl$	

Het geval $zsl = msl$ betekent uiteraard beëindiging van het zoekproces met het resultaat dat het gevraagde record is gevonden.

Als $zsl < msl$, resp. $zsl > msl$, wordt de juist beschreven werkwijze herhaald op het 'onderste' resp. het 'bovenste' deel van de recordverzameling. Dit proces wordt net zo lang herhaald totdat of het gevraagde record is gevonden of de te onderzoeken deelverzameling van records leeg is.

Wij moeten verder nog onderscheiden een intern zoekproces (op een verzameling records in het interne geheugen) en een extern zoekproces (op een verzameling records in een extern geheugen). Wij

zullen hier een programmaschets geven voor een extern zoekproces (zie de opgaven voor een intern zoekproces).

Voor de programmaschets zullen wij veronderstellen dat

- het aantal records bekend is en via de opdracht accept(n) de variabele n dit aantal zal aangeven;
- elk record precies een adresplaats beslaat. Als een record het adres a heeft, hebben de 'buren' de adressen a-1 en a+1.

Bij het programma gebruiken we verder nog de variabelen la, ha, ma om aan te geven: laagste, hoogste en middelste aadres.

```

begin type record strec(sl,data);
      file stbest of strec
endtype;
var stb: stbest endvar;

ready(stb);

begin var sr: strec;
      zsl,la,ha,ma: integer;
      gevonden: boolean
endvar;
accept(zsl); accept(ha);
la := 1; gevonden := false;
while not gevonden and la < ha
  do ma := (la+ha) div 2;
      read(stb,ma,sr);
      if sr.sl = zsl
        then gevonden := true
        else if zsl < sr.sl
          then ha := ma-1
          else la := ma+1
        fi
      fi
    od;
if gevonden
  then print(sr)
  else print("niet aanwezig")
fi
end

```

Het aantal vergelijkingen zal bij binair zoeken maximaal $2^{\log n + 1}$ bedragen, dus (voor grote waarden van n) beduidend minder dan het gemiddelde aantal bij lineair zoeken. Dit verschil is met name bij extern zoeken belangrijk omdat daar voor elke vergelijking een (relatief tijdrovende) leesopdracht nodig is.

Ad c. (geordende kettingstructuur)

Na het bovenstaande is wel duidelijk dat in dit geval alleen de lineaire zoekmethode kan worden toegepast.

OPGAVEN

Sorteren

- 5.1. Ga na hoe de invoegmethode, de selectiemethode en de interne straight two-way merge sort werken op een dalende rij getallen, als gevraagd wordt deze rij oplopend te sorteren.
- 5.2. Geef een programmaschets voor de interne straight two-way merge sort.
- 5.3. Een verzameling records is extern in de hier gegeven volgorde van sleutelwaarden opgeslagen: 803, 612, 233, 942, 876, 375, 648, 226, 232, 423, 171, 394, 431, 594, 015, 791. Sorteert deze verzameling met behulp van een
 - a. 2- en 4-way merge;
 - b. polyphase merge met 3 bandeenheden.
- 5.4. Beschrijf het sorteerproces bij externe sortering op een adresseerbaar achtergrondgeheugen, als gegeven is dat de tabel met de paren (sleutelwaarde, adres) niet in zijn geheel in het interne geheugen kan worden opgenomen.

Zoeken

- 5.5. Geef een programmaschets voor het zoeken van een record in een ongeordende consecutieve structuur.
- 5.6. Geef een programmaschets voor het intern binair zoeken.
- 5.7.
 - a. Bepaal de gemiddelde lengte van het zoekproces voor binair zoeken op een aantal van 15 records.
 - b. Idem voor een aantal van 20 records.
 - c. (Voor de liefhebbers) Toon aan dat de gemiddelde lengte Z_n bij binair zoeken gelijk is aan:

$$Z_n = m - 1 + \frac{2k+m}{2^m+k-1} \quad \text{als het aantal records gelijk is aan}$$

$$n \text{ en } n = 2^m - 1 + k \quad (m \geq 1 \text{ en } 0 \leq k < 2^m).$$

- 5.8. a. Aan welke voorwaarden voor opslag en fysieke structuur moet een verzameling records voldoen om binair zoeken op deze verzameling mogelijk te maken?
- b. Waarom is in een geordende kettingstructuur alleen lineair zoeken mogelijk?

6 DIRECTE BESTANDSORGANISATIE

6.1 INLEIDING

Een bestand is direct georganiseerd als ieder record direct toegankelijk is. Er worden dus, in tegenstelling tot de sequentiële bestandsorganisatie, geen eisen gesteld aan de opslagordening van records. Op het eerste gezicht lijkt de directe organisatie daarom eenvoudiger dan de sequentiële organisatie. We zullen echter zien dat, behoudens uitzonderingsgevallen, de directe organisatie door de eis van directe toegang niet zo eenvoudig is. Deze eis betekent immers dat uit de gegeven sleutelwaarde van een record direct (d.w.z. zonder raadpleging van andere records van het bestand) het adres van dit record kan worden afgeleid.

Ten aanzien van het begrip adres moet onderscheid worden gemaakt tussen absoluut (fysiek) adres en relatief adres. We zullen dit duidelijk maken met een voorbeeld. Stel dat een bestand 4000 records bevat, opgeslagen op een schijvenpakket. Per spoor is plaats voor 4 records, elke cilinder bevat 10 sporen. Voor het totale bestand zijn dan 100 cilinders nodig. Neem aan dat voor dit bestand de cilinders met nummer 67 tot en met 166 worden gereserveerd. Het eerste record van dit bestand zal dan worden opgeslagen op adres 67011 (d.w.z. cilinder 67, spoor 01, plaats 1 op spoor). Het 75e record staat dan op adres 68093. Deze adressen 67011 en 68093 noemt men absolute adressen, terwijl 1 en 75 relatieve adressen zijn, nl. de plaats van het record in het bestand ten opzichte van het begin. Zou men voor opslag de cilinders 25 tot en met 124 nemen, dan zouden de absolute adressen moeten worden gewijzigd, maar niet de relatieve adressen. Tegenwoordig is een gebruiker (meestal) onkundig van de absolute adressen, die door de programmatuur worden toegewezen. Bij een direct toegankelijke opslagstructuur wordt dan ook geen absoluut adres aangegeven, maar wel een relatief adres (meer is ook niet nodig en wenselijk!). Overal waar in dit hoofdstuk gesproken wordt over adres, dient men hieronder te verstaan relatief adres.

Uit de aangeboden sleutelwaarde (de 'symbolic' key) wordt dus het relatieve adres ('actual' key) bepaald en dit wordt vervolgens omge-

zet naar het absolute adres. Op deze laatste omzetting zal in dit boek niet of nauwelijks worden ingegaan. Wel zal verderop uitgebreid ter sprake komen hoe, gegeven de sleutelwaarde, het relatieve adres kan worden bepaald. Hier zij reeds opgemerkt dat deze 'sleutelconversie' plaatsvindt op grond van een eenduidige relatie tussen sleutelwaarde S en (relatief) adres A . Hieruit volgt dat de (veel gebruikte) benaming 'willekeurige (random) bestandsorganisatie' als de directe organisatie wordt bedoeld, eigenlijk misleidend en dus af te raden is. Immers aan een record wordt geen willekeurig, maar juist een berekend adres toegewezen via de eenduidig vastgelegde relatie $A = f(S)$.

Over de consequenties van het bovenstaande nu eerst enkele inleidende opmerkingen:

- allereerst valt op te merken dat voorzover men daartoe de vrijheid heeft, de nodige aandacht dient te worden besteed aan de opbouw van het sleutelveld. Immers dit sleutelveld is de basis voor de directe toegang.
- de eis van directe toegang heeft vérstrekkende consequenties voor de opslagstructuur. We zullen hieraan dan ook een aparte paragraaf wijden (par. 6.2).
- het is zonder meer duidelijk dat alleen met een adresseerbaar geheugenmedium kan worden voldaan aan de eis van directe toegang.

6.2 OPSLAGSTRUCTUUR VOOR DIRECTE BESTANDS-ORGANISATIE

Aangezien de opslagstructuur anders zeer onhanteerbaar zou worden, wordt in een directe bestandsorganisatie vrijwel alleen met vastelengtere records gewerkt. Zoals reeds in het voorgaande werd opgemerkt, worden door de directe bestandsorganisatie geen eisen gesteld aan de opslagordening van de records.

In een sequentieel georganiseerd bestand, bestaande uit 10000 records, is van de voor dit bestand gereserveerde ruimte het eerste tot en met het 10000e adres geheel bezet. Bij een direct georganiseerd bestand daarentegen is het wel of niet bezet zijn van een plaats niet afhankelijk van het aantal records maar van de actuele sleutelwaarden en de gehanteerde sleutelconversie. Om er dan ook zeker van te zijn dat een recordplaats onbezet is, zal bij reservering van de geheugenruimte aan elk sleutelveld de (niet als sleutel voorkomende) waarde ∞ worden toegekend. Een andere (niet als sleutel voorkomende) waarde, zoals bijv. -1, is uiteraard voor dit doel ook geschikt. Via de relatie $A = f(S)$ zal aan elk aangeboden record een plaats in de gereserveerde ruimte worden toegekend. Deze relatie kan verschillende vormen hebben. Wij onderscheiden de

- directe relatie,
- tabelrelatie,
- funktionele relatie.

Elk van deze drie vormen zullen wij nu afzonderlijk bespreken.

Directe relatie. Wij spreken van een directe relatie als het verband tussen adres en sleutelwaarde gegeven is door een lineaire functie van de volgende vorm: $A = (S-C)/K$, waarbij K een natuurlijk getal en C een geheel getal is ≥ 0 . Voor $K=1$ en $C=0$ hebben wij het geval dat het adres gelijk is aan de sleutelwaarde.

Deze relatie is ideaal, als de werkelijk optredende sleutelwaarden zodanig zijn dat de erbij behorende adressen een aaneengesloten rij natuurlijke getallen vormen.

VOORBEELDEN

- de sleutelwaarden 1, 2, 3, ..., 1000 ($K=1$, $C=0$)
- de sleutelwaarden 10001, 10002, 10003, ..., 11000 ($K=1$, $C=10000$)
- de sleutelwaarden 15, 20, 25, ..., 15725 ($K=5$, $C=10$)

In de ideale situatie zullen bij een bestand met n records dus alle adressen vanaf het eerste tot en met het n^e adres van de gereserveerde ruimte aaneengesloten bezet zijn. Dit geeft dus bij een reservering van n adressen 100% bezettingsgraad. Onder bezetting(sgraad) zullen wij verstaan $100 \times$ (werkelijk aantal records/gereserveerde record-plaatsen). Een iets lagere bezetting dan 100% maakt de directe relatie natuurlijk niet onbruikbaar. In het geval van directe organisatie met directe relatie is het bestand uiteraard ook sequentieel georganiseerd (men ga dit na!). Deze organisatievorm is dus erg aantrekkelijk. In de praktijk is zij echter heel weinig toepasbaar. Dit komt omdat actuele sleutelwaarden meestal onregelmatig over een groot traject van waarden verdeeld liggen. Men denke hierbij bijv. aan de grillige waarden van artikelnummers.

VOORBEELD

Een bestand van 10.000 artikelrecords moet met een directe organisatie worden opgezet. De sleutel bestaat uit 10 cijfers en de sleutelwaarden liggen erg willekeurig verdeeld tussen 1 en 10 miljard. Zou nu de directe relatie gekozen worden, dus in dit geval $A=S$, dan zou een ruimte van 10 miljard adressen gereserveerd moeten worden voor een aantal van in feite 10000 records. Dit zou betekenen een miljoen maal meer ruimte dan eigenlijk nodig is!

Aan dit voorbeeld is wel duidelijk dat naarmate de bezettingsgraad afneemt, de directe relatie minder zinvol en zelfs zinloos wordt. In zulke (in de praktijk meest voorkomende) gevallen zullen wij onze toevlucht moeten nemen tot een van de twee andere genoemde relaties.

Tabelrelatie. Nu ligt het verband tussen sleutelwaarde en adres opgeslagen in een apart hulpbestand, dat wij het tabelbestand (table file) zullen noemen, in tegenstelling tot het gegevensbestand (data file), waarin de eigenlijke gegevens liggen opgeslagen. Van elk record in het gegevensbestand is dus het paar (sleutelwaarde, adres) in het tabelbestand opgenomen, waarbij dan adres het adres bevat van het record in het gegevensbestand.

VOORBEELD

tabelbestand		gegevensbestand	
sleutel	adres	sleutel	overige recordvelden
1705	5	1833	-----
1821	2	1821	-----
1833	1	1902	-----
1902	3	3273	-----
2051	6	1705	-----
3273	4	2051	-----
enz.	enz.	enz.	enz.

Voor dit hulpbestand moet apart de organisatievorm worden vastgesteld. Uiteraard dient dit te geschieden op basis van de doelstelling van dit hulpbestand. Het doel is nl. een gunstige bezettingsgraad van (de geheugenruimte voor) het gegevensbestand, waarbij de directe toegankelijkheid van het gegevensbestand gewaarborgd moet blijven.

Voor het tabelbestand is de sequentiële organisatie de aangewezen organisatievorm. Het gegevensbestand moet zich echter, zoals we reeds zagen, op een adresseerbaar medium bevinden. Het ligt alsdan voor de hand het bijbehorende tabelbestand op hetzelfde medium op te slaan. Bovendien geeft opslag op een adresseerbaar medium enig voordeel bij het onderhoud (geen volledig nieuw tabelbestand nodig).

Stel dat een bestand, dat direct georganiseerd moet worden met tabelrelatie, het aantal records (op een gegeven moment) gelijk is aan 8723 en dat een record van het bijbehorende tabelbestand 500 paren (sleutel, adres) kan bevatten. Dan zal het tabelbestand uit 18 records bestaan, waarbij het laatste record dan 'niet helemaal vol' zal zijn. Wij zullen afspreken dat in zo'n geval het tabelrecord na het laatste paar, nog een fictief paar heeft met sleutelwaarde ∞ , (en adres ongedefinieerd).

In ons voorbeeld zal de sleutelwaarde van het 224e paar in het 18e record gelijk ∞ zijn. Voor het lezen van het gehele tabelbestand zijn in dit geval 18 leesopdrachten nodig. Het zoeken en lezen van een willekeurig record, gegeven de sleutelwaarde, zal dus gemiddeld $10\frac{1}{2}$ leesopdrachten vergen ($9\frac{1}{2}$ tabelrecords plus het gevraagde

record uit het tabelbestand*). Het werkelijke aantal leesopdrachten zal minimaal 2 en maximaal 19 zijn. Bij zo'n aantal van 19 leesopdrachten wordt het begrip directe toegankelijkheid toch wel enig geweld aangedaan. Het aantal leesopdrachten, nodig voor het zoeken van een willekeurig record, kan sterk worden gereduceerd door het tabelbestand 'hiërarchisch' op te bouwen. Aan de tabelrecords wordt dan een hoofdtabel toegevoegd. Deze hoofdtabel bevat de hoogste sleutelwaarde voor ieder tabelrecord. In ons voorbeeld zou deze hoofdtabel dan bestaan uit 18 sleutelwaarden, plus een afsluitende (niet voorkomende) sleutelwaarde ∞ .

De plaats van een hoogste sleutelwaarde in de hoofdtabel geeft meteen het relatieve adres van het tabelrecord in het tabelbestand. Met een hoofdtabel is voor het zoeken en lezen van een willekeurig record het aantal leesopdrachten teruggebracht tot (een constante waarde van) 3. (Men ga dit na!) Mocht de hoofdtabel (door de omvang van het gegevensbestand) te groot worden (voor de maximumgrootte van de inleesbuffer), dan doet men er verstandig aan boven deze hoofdtabel een 'nieuwe hiërarchische laag' aan te brengen in de vorm van een super-hoofdtabel, enz. Voor een omvangrijk bestand kan volgens deze opzet het tabelbestand een hiërarchische opbouw van enkele lagen diep hebben. (Het toekomstige Viditelsysteem zal hierop berusten.) Uiteraard zal een hiërarchische opbouw meer werk betekenen bij onderhoud van het bestand, maar dit meerwerk zal in het algemeen zeer behoorlijk worden gecompenseerd door de drastische vermindering van het (gemiddelde) aantal leesopdrachten.

Met de tabelrelatie kan voor de ruimte, gereserveerd voor het gegevensbestand, een zeer gunstige bezettingsgraad worden bereikt. De bezettingsproblemen worden als het ware 'afgewenteld' op het tabelbestand. Vrijgekomen plaatsen (door voorgaande verwijderingen) in het gegevensbestand, kunnen gemakkelijk weer worden bezet met nieuw in te voegen records. Reorganisatie van het gegevensbestand is dan ook eigenlijk nooit nodig. Hoogstens kan het voorkomen dat voor het gegevensbestand een grotere ruimte moet worden gereserveerd, nl. als het totale aantal records groter wordt dan het totale aantal beschikbare recordplaatsen in de ruimte voor het gegevensbestand**).

Funktionele relatie. Hiermee wordt bedoeld dat via een functievoorschrift het verband tussen adres en sleutelwaarde is vastgelegd. Eigenlijk zijn de directe relatie en tabelrelatie bijzondere gevallen van een functie die aan elke waarde van S een waarde van A

*) In de bepaling van dit gemiddelde zit een (verwaarloosbare) fout vanwege het feit dat het laatste tabelrecord niet geheel gevuld is.

**) Dit betekent in feite uitbreiding van het aantal relatieve adressen, hetgeen fysiek een vrij ingewikkelde aanpassing (elders in het geheugenmedium) kan betekenen. Doch, zoals gezegd, wij zullen ons hoofdzakelijk beperken tot de behandeling van relatieve adressen.

toevoegt. Uit het verdere verloop zal echter blijken dat het zinvol is de eerste twee relatievormen te onderscheiden van de funktionele relatie. Wij zullen het begrip funktionele relatie dan ook verder alleen in engere zin gebruiken, nl. met uitsluiting van directe en tabelrelatie.

We zullen de funktionele relatie nu eerst aan de hand van een voorbeeld bespreken.

VOORBEELD

Gegeven is een artikelbestand van 2000 records. De mogelijke sleutelwaarden zijn willekeurig verspreid tussen 0 en 10 miljoen. Het sleutelveld bestaat dus uit 7 cijferposities. Het bestand moet direct worden georganiseerd met funktionele relatie. Voor dit laatste wordt het volgende gekozen. Van een gegeven sleutelwaarde wordt eerst een getal van 4 cijfers (g_4) gevormd door achtereenvolgens het 2e, 6e, 5e en 3e cijfer (van rechts gerekend) van de sleutel te nemen als 4e, 3e, 2e en 1e cijfer (van rechts) van g_4 . Het recordadres wordt nu verkregen door de rest te nemen na deling van g_4 door 2500. Is bijv. de sleutelwaarde 4611060 gegeven, dan is $g_4 = 6610$ en het recordadres is $6610 \bmod 2500 = 1610$. Uiteraard is deze conversie van sleutelwaarde naar adres in een formeel rekenvoorschrift vast te leggen en wel als volgt:

```
function procedure adres(s: integer): integer;
  begin var C2,C3,C5,C6: integer
        endvar;
        C2 := (s mod 100) div 10;
        C3 := (s mod 1000) div 100;
        C5 := (s mod 100000) div 10000;
        C6 := (s mod 1000000) div 100000;
        adres := (C2 * 1000 + C6 * 100 + C5 * 10 + C3) mod 2500
  end;
```

Bij deze funktionele relatie zijn dus 2500 adressen beschikbaar. Deze beschikbare capaciteit van 2500 recordplaatsen laat dus nog een uitbreiding van 500 records toe op het huidige bestand van 2000. Op het eerste gezicht lijkt deze uitbreidingsmogelijkheid (bij niet al te hoge file-turnover) vrij ruim. We moeten echter bedenken dat, in tegenstelling tot de organisatie met de tabelrelatie, het gegevensbestand met opeenvolgend aangeboden records 'kriskras' wordt gevuld. Hierbij is het best mogelijk dat sommige adressen (voorlopig) niet en andere adressen meer dan eens aan bod komen. Zo zal de boven beschreven funktionele relatie voor de records met sleutel 4611060 resp. 0118042 hetzelfde adres 1610 aanwijzen. Sleutels die tot eenzelfde adres leiden noemt men 'synoniemen'. Het ligt voor de hand dat men synoniemen zoveel mogelijk tracht te vermijden.

Uiteraard kan hier soelaas worden geboden door de beschikbare geheugenruimte uit te breiden. Als men bijvoorbeeld in bovenstaand geval 5000 adressen zou reserveren (en dus zou delen door 5000 in plaats van 2500), zouden de records met sleutel 4611060 resp. 0118042 leiden tot het adres 1610 resp. 4110. Genoemde synoniemen zouden dan zijn verdwenen, echter ten koste van extra geheugenruimte. In feite hebben we hier dus te maken met twee 'tegengestelde kostenfactoren'. Immers, terugdringen van de 'kostenfactor' geheugenruimte zal de 'kostenfactor' synoniemenbeheer opjagen en omgekeerd. In de volgende paragrafen wordt op dit probleem uitgebreid ingegaan.

Uit het voorgaande is wel reeds duidelijk dat de funktionele relatie (men spreekt ook wel van 'hash-adressing') zo gekozen moet worden dat bij een gegeven verzameling sleutelwaarden de bezettingsgraad economisch aanvaardbaar is en dat ook niet teveel last wordt ondervonden van synoniemen. Op verschillende 'conversiealgoritmen' wordt in de volgende paragraaf nader ingegaan.

Betreffende de 'kostenfactor' synoniemenbeheer de volgende toelichting. De ruimte van potentieel voorkomende sleutels wordt op de (vrijwel altijd) kleinere ruimte van beschikbare recordadressen afgebeeld. Daarom moet een voorziening worden getroffen voor de ('botsings'-)mogelijkheid dat uit twee verschillende sleutels (synoniemen) hetzelfde recordadres volgt. Als regel berust deze voorziening op een van de volgende methoden.

- a. Buiten het in eerste instantie voor het bestand ter beschikking gestelde 'primaire' gebied (prime area) wordt een 'overloop'gebied (overflow area) gereserveerd, dat gebruikt zal worden voor opslag van records die hun primaire plaats bezet vinden. De opslag van 'overflow records' kan op verschillende manieren geëffectueerd worden, bijv. ongeordend met een koppeling van synoniemen in primair en overloopgebied door middel van een kettingstructuur.
- b. Men reserveert voor iedere getransformeerde sleutel primair niet één maar meer recordposities. Deze recordposities vormen tezamen één 'bin' of 'bucket', waarin evenzovele synoniemen kunnen worden opgeslagen. Dit aantal mogelijk op te slaan synoniemen in een bin noemen we de bingrootte. Eigenlijk is geval a een bijzonder geval van b, nl. voor bingrootte 1.
- c. Een overlooprecord wordt ondergebracht in bijvoorbeeld de eerstvolgende vrije lokatie, volgende op de lokatie die in eerste instantie was berekend. In dit geval maakt men geen onderscheid tussen primair en overflowgebied. Men kan (terwille van een snel terugzoeken) bovendien nog een kettingstructuur gebruiken voor het koppelen van synoniemen.

De sub a en b genoemde methoden zullen uitgebreid worden besproken in paragrafen 6.3 en 6.4. In paragraaf 6.5 zal nader op geval c worden ingegaan. Hier zij reeds opgemerkt dat geval c van toepassing

kan zijn bij relatief kleine records, die met velen tegelijk in een grote inleesbuffer kunnen worden ingelezen.

Samenvattend kan over de verschillende vormen van de relatie

$A = f(S)$ worden gesteld:

- directe relatie is (economisch) alleen haalbaar als de voorkomende sleutelwaarden gelijkmatig en dicht verdeeld zijn over het interval, begrensd door de grootst mogelijke en kleinste mogelijke sleutelwaarde. Wordt aan deze voorwaarde niet voldaan (zoals in de praktijk meestal het geval is), dan is de bezettingsgraad van het geheugen-medium onverantwoord laag.
- tabelrelatie en funktionele relatie zijn in principe altijd toepasbaar. De tabelrelatie geeft een zeer gunstige bezettingsgraad voor het gegevensbestand, echter ten koste van een apart tabelbestand en dus een hoeveelheid extra ruimte en programmatuur. De tabelrelatie is (vanwege het zoeken in het hulpbestand) 'minder direct' dan de funktionele relatie. Deze heeft echter weer het nadeel van een minder gunstige bezettingsgraad. Als deze bezettingsgraad hoog wordt gehouden, gaat dit ten koste van meer synoniemen en dus meer overflowruimte.

Uit het voorgaande is duidelijk dat een direct georganiseerd bestand bij directe relatie sequentieel kan worden verwerkt. Ook bij tabelrelatie kan dit nog op enigszins redelijke wijze. Hierbij zal de niet-sequentiële ordening in het gegevensbestand als regel de gemiddelde toegangstijd per record nadelig beïnvloeden. Bij de funktionele relatie hebben we een uitsluitend direct toegankelijk bestand. Immers het bestand zelf kan niet sequentieel worden verwerkt (tenzij het hele bestand eerst wordt gekopieerd en daarna gesorteerd). Het is namelijk vooraf niet bekend welke sleutelwaarden voorkomen en opeenvolgende adressen zullen over het algemeen geen opeenvolgende sleutelwaarden bevatten!

Een en ander is ook nog eens schematisch in onderstaande tabel weergegeven.

vorm van $A = f(S)$	bezettings- graad gegevens- bestand	hulpbe- stand(en)	overflow ruimte	sequentieel verwerkbaar
direkte relatie	100% (in ideale geval)	neen	neen	ja
tabel relatie	100%	ja	neen	(aarzelend) ja
funktionele relatie	< 100%	neen	ja	neen

6.3 SLEUTELCONVERSIE BIJ FUNKTIONELE RELATIE

Sleutelconversie geeft geen problemen bij directe en tabelrelatie, echter wel bij de funktionele relatie. De voorkomende sleutelwaarden zijn weliswaar ongelijk, maar de verdeling van deze waarden over het beschikbare gebied is als regel allesbehalve uniform. Immers door de manier waarop de sleutels worden opgebouwd zijn vaak al lang voordat over het efficiënt geheugengebruik gedacht wordt, gaten en ophopingen ontstaan. Meestal volgt een sleutelwaarde uit klassifikatie-overwegingen, waarbij aan bepaalde onderdelen een bepaalde betekenis wordt toegekend. Door de aard van de te klassificeren grootheden kunnen bepaalde cijfercombinaties dan niet voorkomen ('gaten'), terwijl andere cijfercombinaties wellicht vaak voorkomen ('ophopingen'). De te kiezen methode voor sleutelconversie moet nu zo zijn dat de geconverteerde sleutels zo homogeen mogelijk verdeeld zijn. Immers, met een homogene verdeling van de geconverteerde sleutels kan een redelijk hoge bezettingsgraad worden bereikt zonder dat dit ten koste gaat van een groot aantal synoniemen en dus een grote mate van overflow.

Men kan zich indenken dat zo'n sleutelconversiemethode niet eenvoudig te vinden is voor een eenmaal historisch gegroeid sleutelsysteem, waarvan de statistische eigenschappen vaak niet of nauwelijks bekend zijn. Het eenvoudigste ligt de zaak als de sleutelwaarden uniform (homogeen) verdeeld zijn. Deze verdeling van sleutelwaarden komt echter, omdat zij geen klassificerend karakter heeft, in de praktijk zelden voor (toevallig wel bij de in ontwikkeling zijnde landelijke bevolkingsadministratie). Niet-homogene verdelingen kunnen bijzonder gevoelig zijn voor bepaalde conversiemethoden in de zin van een relatief hoog aantal synoniemen.

Veel methoden voor sleutelconversie zijn in de loop der tijd langs empirische weg ontwikkeld:

- 'truncation', d. w. z. het weglaten van (veelal de voorste) cijfers van een sleutel en het gebruik van de overblijvende cijfers als nieuwe sleutel. Door het klassificerend karakter van een sleutel is deze methode als regel, ondanks zijn eenvoud, niet aan te raden zonder de oorspronkelijke sleutelverzameling zorgvuldig te onderzoeken; de straf kan immers zijn dat na conversie sterke ophopingen ontstaan.
- 'extraction', d. w. z. bepaalde cijfers van de oorspronkelijke sleutel worden, achter elkaar geplaatst, als nieuwe sleutel opgevat (als regel zal men cijfers waarin sterke ophopingen voorkomen hierbij uiteraard niet meenemen). Ook deze methode mag pas na een zorgvuldig onderzoek van de oorspronkelijke sleutelverdeling gehanteerd worden (zie voorbeeld in par. 6.2).
- 'folding' is meestal een beter proces dan truncation omdat hierbij in ieder geval de volledige oorspronkelijke sleutelwaarde gebruikt wordt. Deze wordt daartoe nl. gesplitst in twee of meer stukken die

dan bij elkaar opgeteld worden, bijv. voor folding van 6 naar 2 cijfers:

$$\begin{array}{cccc} 12 & 34 & 56 & 02 \\ 63 & 72 & 88 & 23 \end{array} \quad \begin{array}{l} \text{(immers } (12+34+56) \text{ modulo } 100 = 2) \\ \text{(immers } (63+72+88) \text{ modulo } 100 = 23) \end{array}$$

Opmerking: Soms wordt onder folding ook verstaan het berekenen van de nieuwe sleutel door middel van een aritmetische berekening op sleuteldelen. Folding kan goed werken, maar heeft enigszins een hocus-pocus karakter en vereist eveneens een statistisch onderzoek van de oorspronkelijke distributie van sleutels. (Nullen in de oorspronkelijke sleutels hebben een slecht effect!)

- 'mid-squaring' is een andere met voorzichtigheid te hanteren methode om random getallen te berekenen: een sleutelwaarde wordt met zichzelf vermenigvuldigd waarna bijv. de 'middelste' n cijfers als nieuwe sleutel gehanteerd wordt. Mid-squaring kan echter veel nullen produceren en is in het algemeen daarom niet aan te raden.
- 'radix-conversion' werd vroeger, toen delingsbewerkingen op een computer tamelijk langzaam waren, nog wel eens gebruikt. Bijv. voor radix 10 \rightarrow radix 11 conversie, gevolgd door truncatie, als volgt:

$$1234 : 1 \times 11^3 + 2 \times 11^2 + 3 \times 11^1 + 4 = 1610 \rightarrow 10.$$

- 'prime division'. Wanneer n adressen beschikbaar zijn (dus n verschillende nieuwe sleutelwaarden mogelijk zijn), dan kan de rest van de deling van de oorspronkelijke sleutelwaarde door n als nieuwe sleutelwaarde gebruikt worden. Een eigenschap van deling is dat n opeenvolgende originele sleutels ook n verschillende opeenvolgende geconverteerde sleutelwaarden geven. Dit is ook nog waar als de oorspronkelijke sleutelwaarden telkens een bedrag d van elkaar verschillen, mits d en n geen gemeenschappelijke delers bezitten. Bijvoorbeeld voor d = 3 en n = 7:

$$\text{sleutelwaarden} \left\{ \begin{array}{l} \text{voor conversie: } 20, 23, 26, 29, 32, 35, 38, 41, \text{ enz.} \\ \text{na conversie: } 6, 2, 5, 1, 4, 0, 3, 6, \text{ enz.} \end{array} \right.$$

Aan de voorwaarde dat d en n geen gemeenschappelijke delers bezitten, is uiteraard altijd voldaan als n een priemgetal is. In de praktijk zullen sleutelwaarden wel niet equidistant verdeeld zijn, maar toch ligt het voor de hand om voor de deler steeds een priemgetal te gebruiken en wel (teneinde de beschikbare nieuwe ruimte zo goed mogelijk te gebruiken) het grootste priemgetal dat net nog kleiner is dan het aantal beschikbare adressen (nieuwe sleutelwaarden).

Priemgetaldeling is de veiligste methode voor sleutelconversie, wanneer men geen tijd heeft voor een analyse van oorspronkelijke sleutelwaarden, maar natuurlijk niet noodzakelijk de beste methode voor iedere verdeling van sleutelwaarden. Niet alle priemgetallen zijn even goed te gebruiken; i. h. b. kunnen priemgetallen van de vorm $a \times 10^b \pm 1$ voor kleine a waarden beter vermeden worden, dus priemgetallen als 11, 19, 29, 31, 41, 101, 199, 401, 499, 599, 601, 701, 1999, 2999, 3001, 4001, 4999, 7001, 8999, 9001, 49999, 59999, 70001, 79999, 900001, enz. Uit de ontwikkeling $(p \pm 1)^{-1} = 1/p \pm 1/p^2 \pm 1/p^3$ zien we met $p = 10^b$ dat gebruik van deze priemgetallen neerkomt op optelling van b -digit groepjes van de oorspronkelijke sleutel, d.w.z. op folding. Uit ervaring is bekend dat folding echter meestal minder goed is dan prime-division. Ook als $a \neq 1$ is komt gebruik van deze priemgetallen vrijwel op folding neer.

De bespreking van sleutelconversie wordt nu besloten met een wat uitgebreider (overigens sterk gestileerd) voorbeeld. Gegeven is een bestand met 100 records. De sleutel bestaat uit drie cijfers. Tabel 1 bevat sleutelwaarden, welke kunnen zijn ontstaan door een aselechte trekking uit een uniforme verdeling tussen 0 en 1000. (In verband met verder gebruik van dit voorbeeld zijn de sleutelwaarden genummerd; deze nummering heeft verder geen betekenis.) Het bestand moet direct worden georganiseerd met een funktionele relatie. Voor deze relatie kiezen we de delingsvorm, dus: relatief adres = sleutelwaarde mod n , waarbij n verderop wordt bepaald.

Bestandsgroei wordt buiten beschouwing gelaten. We zullen aannemen dat voor het primaire gebied 100 recordposities beschikbaar zijn. Hiervan uitgaande zullen we verder twee varianten beschouwen.

- a. Per relatief adres in het primaire gebied kan maximaal één record worden opgeslagen. Bij sleutelconversie zal dus een aantal van twee of meer synoniemen per adres tot overflow leiden.
- b. Per relatief adres in het primaire gebied kunnen maximaal twee records worden opgeslagen. Bij sleutelconversie zullen nu drie of meer synoniemen per adres tot overflow leiden.

We zullen nu voor beide varianten de funktionele relatie nader preciseren en zien wat verder de resultaten zullen zijn wat betreft de overflow.

Ad a. Zouden we priemdeling toepassen, dan zou in dit geval het priemgetal 97 als deler fungeren. Aangezien echter de sleutelwaarden uniform verdeeld zijn, kan een niet-priemgetal even goed als deler fungeren. Eenvoudigheidshalve hebben we hier 100 als deler genomen. Dus de funktionele relatie is: relatief adres = sleutelwaarde mod 100. Deze relatie, toegepast op tabel 1, geeft de uitkomsten als weergegeven in tabel 2 (kolom F_1 ; nagaan!). Het blijkt dus dat, ofschoon de primaire ruimte in principe voldoende plaats biedt voor het bestand, er toch 31 records in de overflow terechtkomen (nagaan!).

Tabel 1: Honderd sleutelwaarden

nr.	sleutel- waarde	nr.	sleutel- waarde	nr.	sleutel- waarde	nr.	sleutel- waarde
1	710	26	354	51	796	76	181
	230		695		363		700
	355		068		765		683
	508		552		779		348
	465		758		323		439
6	521	31	303	56	828	81	562
	131		908		482		601
	306		404		297		349
	904		057		686		659
	282		277		555		174
11	605	36	632	61	006	86	267
	361		086		449		080
	707		113		280		197
	608		635		518		172
	186		456		245		649
16	116	41	856	66	077	91	070
	673		885		901		334
	987		520		143		746
	694		327		041		882
	630		781		780		895
21	607	46	481	71	926	96	421
	615		333		193		223
	573		873		190		291
	749		660		388		509
	972		167		741	100	329

Ad b. Analoog aan geval a wordt hier voor de funktionele relatie aangehouden: adres = sleutelwaarde mod 50. Toegepast op tabel 1 krijgen we de resultaten als weergegeven in kolom F₂ van tabel 2. Het aantal records in de overflow is nu 24 (nagaan!).

Veronderstellen we echter dat voor de primaire ruimte 200 recordposities beschikbaar zijn, en de varianten c resp. d analoog aan a resp. b (dus in variant c resp. d maximaal een resp. twee records per adres), dan krijgen we:

Tabel 2: Frequenties (F_1 resp. F_2) van voorkomende adressen door toepassing op tabel 1 van:

(F_1) : adres = sleutelwaarde $\text{mod } 100$

(F_2) : adres = sleutelwaarde $\text{mod } 50$

adres	F_1	F_2	adres	F_1	F_2	adres	F_1	adres	F_1
0	1	1	25	0	0	50	0	75	0
1	2	2	26	1	1	51	0	76	0
2	0	1	27	1	3	52	1	77	2
3	1	1	28	1	1	53	0	78	0
4	2	3	29	1	2	54	1	79	1
5	1	3	30	2	5	55	2	80	3
6	2	4	31	1	4	56	2	81	3
7	2	3	32	1	4	57	1	82	3
8	3	4	33	1	2	58	1	83	1
9	1	2	34	1	1	59	1	84	0
10	1	2	35	1	2	60	1	85	1
11	0	1	36	0	3	61	1	86	3
12	0	1	37	0	1	62	1	87	1
13	1	2	38	0	1	63	1	88	1
14	0	0	39	1	1	64	0	89	0
15	1	3	40	0	1	65	2	90	1
16	1	1	41	2	3	66	0	91	1
17	0	2	42	0	0	67	2	92	0
18	1	2	43	1	2	68	1	93	1
19	0	0	44	0	1	69	0	94	1
20	1	2	45	1	3	70	1	95	2
21	2	2	46	1	2	71	0	96	1
22	0	2	47	0	2	72	2	97	2
23	2	5	48	1	1	73	3	98	0
24	0	1	49	4	4	74	1	99	0

Ad c. met de funktionele relatie: adres = sleutelwaarde $\text{mod } 200$, een aantal van 15 records in de overflow (nagaan!).

Ad d. met de funktionele relatie: adres = sleutelwaarde $\text{mod } 100$, een aantal van 8 records in de overflow (nagaan!).

Vergelijking van de gevallen a, b, c en d bevestigt de overigens voor de hand liggende, vermoedens:

- vergroting van de primaire ruimte leidt tot minder overflow records.

- vergroting van de bingrootte (= maximaal aantal records per adres) leidt eveneens tot minder overflow records.

In de volgende paragraaf wordt de theoretische basis van deze vermoedens blootgelegd en, wat belangrijker is, daarmee tevens het overflow probleem kwantitatief benaderd.

6.4 OVERFLOW BIJ EEN FUNKTIONELE RELATIE

Bij directe bestandsorganisatie met een funktionele relatie (dus met sleutelconversie) is overflow in het algemeen niet te vermijden. Aan de overflow-problematiek zijn twee aspecten te onderscheiden:

- A. het bepalen van de mate van overflow, gegeven de detailspecificaties van de organisatie en de verdeling van de sleutelwaarden
- B. de wijze van opslaan van overflow records en een geschikte fysieke structuur. Hierbij gaat het erom de nadelen van overflow zo klein mogelijk te maken.

A. BEPALING VAN DE MATE VAN OVERFLOW

Eerst enkele afspraken over definities en notaties:

- g : toegewezen geheugenruimte (primaire gebied) voor het bestand, uitgedrukt in aantal recordposities.
- m : totaal aantal records in het bestand
- f : vullingsgraad (load factor) van het primaire gebied. Hieronder wordt verstaan de verhouding tussen het totaal aantal records in het bestand en het totaal aantal beschikbare recordplaatsen in het primaire gebied. Anders gezegd: de vullingsgraad geeft aan welke fractie van het primaire gebied gevuld zal zijn als het gehele bestand in dit primaire gebied wordt opgeslagen. Dus $f = m/g$.
- b : aantal recordposities per bin (bingrootte, binsize),
- μ : gemiddeld aantal records per bin als het gehele bestand in het primaire gebied is opgeslagen. Dus $\mu = m/(g/b) = fb$.

Verder veronderstellen we dat de funktionele relatie zodanig is gekozen dat de geconverteerde sleutels homogeen verdeeld zijn. Onder deze veronderstelling heeft bij conversie van een willekeurige sleutel dus elk binadres een even grote kans om als geconverteerde sleutel op te treden. Deze kans is $1/(\text{aantal bins}) = b/g$. Het aantal records n dat bij conversie aan een bepaald binadres wordt toegewezen, is binomiaal verdeeld. Dus

$$p_n = P(\underline{n} = n) = \binom{m}{n} (b/g)^n (1-b/g)^{m-n} \quad (1)$$

In praktisch voorkomende gevallen is m groot genoeg en b/g voldoende klein om de genoemde binomiale verdeling zeer goed te kunnen bena-

deren met de Poissonverdeling, dus

$$p_n \approx \frac{\mu^n}{n!} e^{-\mu} \quad (2)$$

Opmerking: (2) kan als volgt uit (1) afgeleid worden. Als $m(b/g)$, dus μ constant blijft geldt (voor elke vaste waarde n):

$$\lim_{m \rightarrow \infty} \binom{m}{n} (b/g)^n (1-b/g)^{m-n} =$$

$$\lim_{m \rightarrow \infty} \frac{m!}{n! (m-n)!} \left(\frac{\mu}{m}\right)^n \left(1 - \frac{\mu}{m}\right)^{m-n} =$$

$$\lim_{m \rightarrow \infty} \frac{\mu^n}{n!} \cdot \left(1 - \frac{\mu}{m}\right)^m \cdot \frac{m}{m} \cdot \frac{m-1}{m} \cdots \frac{m-n+1}{m} \left(1 - \frac{\mu}{m}\right)^{-n} =$$

$$\frac{\mu^n}{n!} e^{-\mu}.$$

Gegeven de bingrootte b en de vullingsgraad f is nu het gemiddelde aantal overflow records C_b per bin te bepalen als functie van b en f , dus ook van b en μ :

$$C_b = \sum_{n=b+1}^{\infty} (n-b) p_n = \quad (3)$$

$$= \sum_{n=b+1}^{\infty} n p_n - b \sum_{n=b+1}^{\infty} p_n =$$

$$= \sum_{n=b+1}^{\infty} n \frac{e^{-\mu} \mu^n}{n!} - b P(\underline{n} \geq b+1) =$$

$$= \mu \sum_{n-1=b}^{\infty} \frac{e^{-\mu} \mu^{n-1}}{(n-1)!} - b P(\underline{n} \geq b+1) =$$

$$= \mu P(\underline{n} \geq b) - b P(\underline{n} \geq b+1) =$$

$$= (\mu - b) P(\underline{n} \geq b) + b P(\underline{n} = b).$$

Het totaal aantal te verwachten overflow records als fractie van het totaal aantal records is:

$$r_b = \frac{C_b \cdot g/b}{m} = \frac{C_b}{\mu} = (1 - \frac{1}{f}) P(\underline{n} \geq b) + \frac{1}{f} P(\underline{n}=b) \quad (4)$$

waarbij n Poisson verdeeld is met gemiddelde $\mu = fb$.

Met behulp van een tabel van de Poissonverdeling is nu op basis van (4) eenvoudig een tabel samen te stellen, waarin het te verwachten overflow-percentage wordt gegeven in afhankelijkheid van bingrootte en vullingsgraad (zie tabel 3). De gevonden percentages voor de gevallen a, b, c en d in het laatste voorbeeld van de vorige paragraaf zijn redelijk in overeenstemming met de tabelwaarden (ga dit na!).

Tabel 3: Verwachte overflow-percentage als functie van bingrootte en vullingsgraad.

bin- grootte	vullingsgraad											
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2
1	4.8	9.4	13.6	17.6	21.3	24.8	28.1	31.2	34.1	36.8	39.4	41.8
2	0.6	2.2	4.5	7.3	10.4	13.7	17.0	20.4	23.8	27.1	30.2	33.3
3	0.1	0.6	1.8	3.6	6.0	8.8	12.0	15.4	18.9	22.4	25.9	29.3
4		0.2	0.8	2.0	3.8	6.2	9.1	12.3	15.9	19.5	23.3	26.9
5		0.1	0.4	1.1	2.5	4.5	7.1	10.3	13.8	17.6	21.4	25.3
6			0.2	0.7	1.7	3.4	5.8	8.8	12.2	16.1	20.1	24.1
7			0.1	0.4	1.2	2.6	4.7	7.6	11.0	14.9	19.0	23.2
8			0.1	0.3	0.8	2.0	4.0	6.7	10.1	14.0	18.2	22.5
9				0.2	0.6	1.6	3.4	5.9	9.3	13.2	17.4	21.9
10				0.1	0.4	1.3	2.9	5.3	8.6	12.5	16.9	21.4
11				0.1	0.3	1.0	2.5	4.8	8.0	11.9	16.3	20.9
12					0.2	0.9	2.2	4.4	7.5	11.4	15.9	20.6
14					0.1	0.6	1.7	3.6	6.7	10.6	15.2	20.0
16					0.1	0.4	1.3	3.1	6.0	9.9	14.6	19.5
18					0.1	0.3	1.0	2.7	5.5	9.4	14.1	19.2
20						0.2	0.8	2.3	5.0	8.9	13.7	18.9
30							0.3	1.2	3.5	7.3	12.3	17.9
40							0.1	0.7	2.6	6.3	11.5	17.5
50							0.1	0.5	2.0	5.6	11.0	17.2
60								0.3	1.7	5.1	10.7	17.0
70								0.2	1.4	4.8	10.4	16.9
80								0.1	1.1	4.5	10.2	16.9
90								0.1	1.0	4.2	10.1	16.8
100								0.1	0.8	4.0	9.9	16.8

Verderop (sub B) zal de opslag van overflow records in het primaire gebied of in een apart overflow-gebied worden besproken. Bij een vullingsgraad groter dan 1 zal altijd een apart overflow-gebied nodig zijn.

Uit tabel 3 kan (door interpolatie) een tabel worden afgeleid, waarin de vullingsgraad wordt gegeven als functie van bingrootte en verwacht overflow-percentage (zie tabel 4). Uit tabel 4 is af te lezen dat een flinke bingrootte, bij gelijkblijvend overflow-percentage, voordeliger is qua primair gebied dan een groot aantal kleine bins. Bijv. bij een overflow-percentage van 10% zal bij een bingrootte van resp. 1, 5 en 10 de vullingsgraad gelijk zijn aan resp. 0.21; 0.79 en 0.94. Overigens moet hierbij wel worden opgemerkt, dat met het toenemen van de bingrootte de toegang tot een record 'minder direct' wordt.

Tabel 4: Vullingsgraad (x 100) als functie van bingrootte en verwacht overflow-percentage

bin-grootte	verwacht overflow-percentage							
	1	3	5	7	10	20	25	30
1			10	15	21	46	61	76
2	13	24	32	39	46	79	94	109
3	24	37	46	54	64	93	107	
4	32	46	55	63	73	101	115	
5	39	53	62	70	79	106	119	
6	44	58	67	74	84	110		
7	48	62	71	78	87	112		
8	52	66	74	81	90	114		
9	55	68	77	83	92	116		
10	57	71	79	85	94	117		
11	60	73	81	87	95	118		
12	62	74	82	89	97	119		
14	65	77	85	91	99	120		
16	68	80	87	93	100			
18	70	82	88	94	101			
20	72	83	90	96	102			
30	78	88	95	99	106			
40	82	91	97	102	107			
50	85	93	99	103	108			
60	87	95	100	104	109			
70	88	96	101	104	109			
80	89	97	101	105	110			
90	90	97	102	105	110			
100	91	98	102	106	110			

Immers, de bin is alleen in zijn geheel direct toegankelijk, maar binnen de bin zal lineair gezocht moeten worden.

B. OPSLAG VAN OVERFLOW RECORDS EN BIJBEHORENDE STRUKTUUR

Zoals reeds gezegd gaat het hier in feite erom de nadelen van overflow zo klein mogelijk te houden. Deze nadelen komen naar voren zowel bij onderhoud als bij gebruik van het bestand. In het algemeen kan worden gesteld dat als het bestand veelvuldig wordt geraadpleegd met daarnaast relatief gering onderhoud, een methode zal worden gezocht waarbij

- het benaderen van een record zo snel mogelijk geschiedt en
- het opzetten en onderhouden van het bestand desnoods een flinke tijd mag vergen (zoals bijvoorbeeld met een kettingstructuur het geval is).

Helaas zijn in de praktijk de frequenties van raadplegingen en mutaties vaak nog niet (voldoende) bekend op het moment dat een bepaalde bestandsorganisatie moet worden gekozen. Over verfijningen, zoals wel dan niet gebruik van kettingstructuren of wel dan niet sequentiële opslag voor overflow records, is dan ook dikwijls moeilijk te beslissen. Aangezien een direct georganiseerd bestand vrijwel altijd op schijf wordt opgeslagen, moet bij het reserveren van een aparte overflow-ruimte met enkele consequenties van de fysieke structuur rekening worden gehouden.

Bij schijven dient men om te beginnen zo mogelijk armbewegingen te vermijden. Overflow-sporen dienen zich dan ook zoveel mogelijk te bevinden op dezelfde cilinder als de primaire sporen. Een overflow vereist dan nog aan extra tijd een (meestal halve) omwentelingstijd (en een overigens te verwaarlozen elektronische schakeltijd van een leeskop).

Wil men nog van de halve omwentelingstijd afkomen dan is te overwegen overflow-ruimte te reserveren aan het 'eind' van ieder primair spoor. Hierbij nemen we wel aan dat apparatuur en programmatuur deze aanpak zinvol maken. Met name moet worden verondersteld dat per leesopdracht een geheel spoor in het werkgeheugen wordt ingelezen.

6.5 OVERLOOP IN VRIJE LOKATIES (SCATTER STORAGE)

In deze paragraaf zullen we nader ingaan op het in 6.2 genoemde geval c, waarbij dus geen onderscheid wordt gemaakt tussen primair en overloopgebied. Als we na adresberekening constateren dat het berekende adres A bezet is, kunnen we proberen het aangeboden record op een vrije plaats onder te brengen. Hiervoor bestaan verschillende

methoden:

- a. 'lineaire' verschuiving: we zoeken p plaatsen ($p > 0$) verder of er een vrije plaats is (de beschikbare plaatsen denken we ons ring-vormig geordend); Het bezwaar van deze methode is de grote kans op mislukte zoekpogingen. Bij een tweede botsing op adres A hebben we nl. een gereede kans dat we pas op $A + 3p$ een vrije plaats vinden, omdat $A + 2p$ al bezet is door een record dat bij een eerste poging op $A + p$ al geen plaats meer vond. Hoewel het toevoegen van een record eenvoudig te programmeren is, geldt dit niet voor het verwijderen van records. Men moet dan eventueel verder verstrooide records terugschuiven of, wat minder werk vergt, het te verwijderen record als zodanig merken.
- b. 'kwadratische' verschuiving: het bezwaar van relatief veel mislukte zoekpogingen wordt ondervangen door bij een botsing $as^2 + bs$ plaatsen verder een lokatie te zoeken. Hierbij zijn a en b (nader te bepalen) constanten en is s het volgnummer van de botsing voor het desbetreffende relatieve adres. In dit geval zullen alleen records die in eerste instantie naar eenzelfde adres worden verwezen, elkaar in de weg zitten.
- c. 'random' verschuiving: bereken bij een botsing met behulp van een (pseudo-)random number generator hoeveel plaatsen verder moeten worden gezocht. Deze generator kan alle getallen in de range $1, \dots, n-1$ produceren. Is de reeks random getallen bijv. 3, 27, 13, 4, ... dan moet dus bij een botsing op adres A achtereenvolgens op de adressen $A+3$, $A+27$, $A+13$, $A+4$, ... naar een vrije lokatie worden gezocht.
- d. 'chaining' methode. Bij de z. g. directe chaining methode worden botsende synoniemen geplaatst op vrije plaatsen en door een ketting verbonden met het oorspronkelijk berekende adres. Zo'n synoniem moet echter verplaatst worden wanneer de lokatie waar hij is ondergebracht later 'opgeëist' wordt door een op die plaats recht hebbend record. Om dit laatste probleem te vermijden is ook de methode van indirecte chaining ontwikkeld, waarbij iedere oorspronkelijke plaats het begin is van een ketting in een apart overflow-gebied.

De efficiëntie van de besproken methoden hangt natuurlijk af van de vullingsgraad f van het beschikbare gebied. Zolang f klein is kan men verwachten dat het 'eerste schot' in het beschikbare gebied raak is. Als echter f tot 1 nadert, zullen meer schoten nodig zijn om een vrije lokatie te vinden. De berekening van de verwachtingswaarde van het aantal schoten is tamelijk gecompliceerd zodat we volstaan met het vermelden van enkele benaderde uitkomsten voor het met succes vinden van een record.

$$E_1 = (1 - \frac{1}{2}f) / (1-f) \quad \text{voor 'lineaire' verschuiving}$$

$$E_2 = -\frac{1}{f} \ln(1-f) \quad \text{voor 'kwadratische' en 'random' verschuiving}$$

$$E_3 = 1 + \frac{1}{2}f \quad \text{voor 'chaining'}$$

Enkele numerieke waarden zijn in bijgaande tabel opgenomen.

gemiddeld aantal benodigde record benaderingen			
f	E_1	E_2	E_3
0.10	1.06	1.06	1.05
0.50	1.50	1.44	1.25
0.75	2.50	1.99	1.38
0.90	5.50	2.56	1.45
0.95	10.5	3.45	1.48
1.00	∞	∞	1.50
2.00	-	-	2.00

Zoals te verwachten, is de lineaire verschuiving de minste van de drie methoden wat zoekfrequentie betreft. De chaining methode komt er het best uit, maar niet vergeten moet worden dat voor de kettingorganisatie extra ruimte nodig is en dat voor $f > 1$ aparte overflowruimte vereist is.

Tenslotte moet erop worden gewezen dat de zojuist besproken technieken alleen maar zinvol kunnen worden toegepast

- bij lage bezettingsgraad van het primaire gebied, en
- relatief lage mutatiefrequentie, en
- relatief kleine recordlengte.

6.6 ONDERHOUD EN GEBRUIK VAN EEN DIRECT GEORGANISEERD BESTAND

Na enkele algemene opmerkingen vooraf zullen wij in deze paragraaf een voorbeeld van wijziging eventueel gevolgd door verwijdering behandelen en wel voor elk van de drie vormen van de relatie $A = f(S)$.

Bij de directe relatie is onderhoud en gebruik relatief eenvoudig; het belangrijkste is dat van elk adres goed wordt bijgehouden of het wel of niet bezet is.

Bij de tabelrelatie is het vooral zaak goed onderhoud te plegen op het tabelbestand. Bij het invoegen van een nieuw record is uiteraard de sleutelwaarde bekend. Het bijbehorende adres moet dan echter nog worden bepaald via een 'administratie' van beschikbare lokaties in het gegevensbestand. Pas na vaststelling van dit adres kan het tabelbestand worden bijgewerkt.

Bij de funktionele relatie zal de snelheid van onderhoud en gebruik in hoge mate afhangen van de kwaliteit van de gebruikte sleutelconversie. Onder kwaliteit dient hier te worden verstaan de mate van homogeniteit in de verdeling der geconverteerde sleutelwaarden. Immers, naarmate deze verdeling meer homogeen is kan men minder synoniemen en dus ook een lagere toegangstijd verwachten. Men spreekt in dit verband dan ook wel van een goede 'randomizer' om aan te geven dat een sleutelconversie een redelijk homogene verdeling oplevert. Het vinden van een goede 'randomizer' is overigens meestal geen eenvoudig probleem. Is deze echter gevonden, dan leveren onderhoud en gebruik geen probleem van betekenis meer en kunnen snel gebeuren. Als wegens de groei van het bestand echter een grotere primaire ruimte moet worden gereserveerd, zal ook een nieuwe 'randomizer' moeten worden bepaald en zullen alle records moeten verhuizen. Bij de funktionele relatie heeft sorteren van mutaties geen enkele zin, omdat de voor elke mutatie benodigde adressering volkomen onafhankelijk van die voor andere mutaties plaatsvindt. Directe organisatie met funktionele relatie is dan ook zeer geschikt voor 'online' toepassingen, al zullen door het bedrijfssysteem voorzieningen getroffen moeten worden om 'bijna gelijktijdige' mutaties op eenzelfde record goed te verwerken.

Nu het aangekondigde voorbeeld.

Gegeven is een direct georganiseerd bestand van uitstaande betalingen. In elk record komen o. a. voor de velden sl (sleutel) en saldo. Verder is gegeven een mutatierecord, waarmee een wijziging van een record in het stambestand bewerkstelligd moet worden. Dit mutatierecord bevat het veld sl (sleutel). Door deze mutatie wordt in het desbetreffende stamrecord eventueel het veld saldo ≤ 0 . Is dit laatste het geval, dan moet het stamrecord worden verwijderd uit het stambestand.

Gevraagd wordt een programmaschets voor de verwerking van bovengenoemd mutatierecord, waarbij rekening dient te worden gehouden met de eventueel optredende fout dat de sleutelwaarde van het mutatierecord niet voorkomt in het stambestand. Is dit laatste het geval, dan moet worden afgedrukt: het mutatierecord en de tekst 'sleutel niet aanwezig'. Kan de mutatie wel worden uitgevoerd, dan dient te worden afgedrukt:

- het stamrecord voor de mutatie,
- het mutatierecord,
- het stamrecord na de mutatie,
- in geval van verwijdering, de tekst: 'record verwijderd'.

Wij zullen dit voorbeeld nu achtereenvolgens behandelen voor de directe relatie, de tabelrelatie en de funktionele relatie. Nadere toelichtingen, die alleen gelden voor een van deze drie gevallen, zullen bij het desbetreffende geval worden gegeven.

A. DIRECTE RELATIE

Wij zullen verder veronderstellen dat:

- A1. het mutatierecord via een acceptopdracht kan worden ingelezen;
- A2. het conversie-algoritme $A = f(S)$ gegeven is door $A = C1 * (S - C2)$ en dat via acceptopdrachten de waarden van $C1$ en $C2$ kunnen worden ingelezen;
- A3. op een adres A , dat een (geldig) record bevat, het sleutelveld uiteraard de waarde S heeft;
- A4. op een adres, dat geen (geldig) record bevat, het sleutelveld de waarde ∞ heeft. Bij verwijdering van een record zal dit veld dus de waarde ∞ moeten krijgen.

De verwerking bestaat nu uit de volgende stappen:

- mutatierecord en $C1$ en $C2$ inlezen
- adres in stambestand bepalen en stamrecord inlezen
- indien geen fout, dan gevraagde verwerking, inclusief eventuele verwijdering, uitvoeren
- indien fout, dan foutmelding.

Onderstaand volgt nu de programmaschets:

```

begin type record strec(sl,saldo,data),
        uprec(sl,data);
        file stbest of strec
endtype;
var stb: stbest endvar;

ready(stb);

begin var sr: strec; ur: uprec;
        adres,C2: integer; C1: real
endvar;
accept(ur); accept(C1); accept(C2);
adres := C1 * (ur.sl - C2);
read(stb,adres,sr);
if sr.sl =  $\infty$ 
then print(ur); print("sleutel niet aanwezig")
else print(sr);          %voor mutatie%
sr := sr  $\oplus$  ur;
print(ur);
print(sr);               %na mutatie%
if sr.saldo  $\leq$  0
then sr.sl :=  $\infty$       %verwijdering voorbereiden%
print("record verwijderd")
fi;
write(stb,adres,sr)

end;

save(stb)

```

end

B. TABELRELATIE

Voor het geval van de tabelrelatie gaan wij uit van de volgende (verdere) veronderstellingen en/of specificaties:

- B1. Het tabelbestand is hiërarchisch opgebouwd met een hoofdtabel en een aantal tabelrecords. Elk tabelrecord bevat ruimte voor 100 paren van sleutelwaarde en adres. Het tabelbestand wordt afgesloten met een paar, waarvan de sleutelwaarde ∞ is. De hoofdtabel bevat de hoogste sleutelwaarde van elk tabelrecord bij creatie of reorganisatie van het bestand. Voor de hoofdtabel zullen wij dezelfde recordstructuur gebruiken als voor een tabelrecord. De hoofdtabel bevindt zich op relatief adres 1 van het tabelbestand. De hoofdtabel kan alleen worden gewijzigd bij reorganisatie van het bestand.
- B2. Het mutatierecord wordt via een acceptopdracht ingelezen.
- B3. Voor het zoeken in de hoofdtabel zullen wij gebruik maken van de (niet nader uitgewerkte) functie procedure hzoek(h, s). Deze procedure kent op basis van de hoofdtabel h aan hzoek het relatieve adres toe van het tabelrecord, waarin de sleutelwaarde s thuisheert. Dit relatieve adres is dan tevens de plaats in de hoofdtabel, waar de hoogste sleutelwaarde van het juist genoemde tabelrecord staat.
- B4. Voor het zoeken in een tabelrecord zullen wij gebruik maken van de (niet nader uitgewerkte) functieprocedure tzoek(tr, s). Deze procedure kent, op basis van het gegeven tabelrecord tr, aan tzoek de plaats toe, waar in dit tabelrecord de sleutelwaarde s staat. Indien deze sleutelwaarde niet in tr voorkomt (in geval dus van een niet te verwerken mutatie), wordt aan tzoek de waarde -1 toegekend.
- B5. Voor het verwijderen van een record zal gebruik worden gemaakt van de volgende procedures:
 - verwtab(tr, i): deze procedure verwijdert uit tabelrecord tr het paar op de i^e plaats
 - voeginvrij(b, a): deze procedure voegt aan de verzameling vrije plaatsen voor bestand b de plaats toe met adres a.

Na het voorgaande zijn wij in staat het mutatieproces te ontwerpen. Dit verloopt in de volgende stappen:

- mutatierecord inlezen
- met behulp van het tabelbestand het adres bepalen van het stamrecord, dat bij het mutatierecord hoort
- indien geen fout dan
 - stamrecord inlezen
 - muteren
 - eventueel verwijderen met aanpassing van tabelbestand
 - inclusief de gevraagde output op de juiste momenten
- indien fout, dan foutmelding.

In het programma zullen wij, behalve van reeds bekende typen en variabelen, ook nog gebruik maken van:

- de typen: paar(sl, verw), tabelrec [1:100] of paar én tabelbest of tabelrec
- de variabelen:
 - tb, van het type tabelbest
 - htr (hoofdtabelrecord) en tr (tabelrecord), beide van het type tabelrec
 - ih (wijzer in hoofdtabelrecord) en it (wijzer in tabelrecord), beide van het type integer.

De programmaschets ziet er nu als volgt uit:

```

begin type record strec(sl,saldo,data),
    uprec(sl,data),
    paar(sl,verw),
    tabelrec[1:100] of paar;
    file stbest of strec,
    tabelbest of tabelrec
endtype;
var stb: stbest; tb: tabelbest
endvar;

ready(stb,tb);

begin var sr: strec; ur: uprec; htr,tr: tabelrec;
    ih,it: integer
endvar;
accept(ur);
read(tb,l,htr);           %hoofdtabel inlezen%
ih := hzoek(htr,ur.sl);   %relatief adres van tabelrecord
                           %bepalen%
read(tb,ih,tr);           %tabelrecord inlezen%
it := tzoek(tr,ur.sl);
if it ≠ -1
    then read(stb,tr[it].verw,sr);
        print(sr)          %vóór mutatie%
        sr := sr ⊕ ur;
        print(ur);
        print(sr);         %ná mutatie%
        if sr.saldo < 0
            then voeginvrij(stb,tr[it].verw);
                verwtab(tr,it);
                write(tb,ih,tr);
                print("record verwijderd")
            else write(stb,tr[it].verw,sr)
            fi
        else print("sleutel niet aanwezig")
        fi
end;

save(stb,tb)

end

```

Opmerking. In bovenstaand programma is geen rekening gehouden met de onwaarschijnlijke situatie dat een tabelrecord door verwijderingen geheel leeg geraakt is.

C. FUNKTIONELE RELATIE

Hiervoor zullen wij de volgende (nadere) veronderstellingen maken en/of specificaties opstellen.

- C1. Het mutatierecord wordt via een acceptopdracht ingelezen.
- C2. De bingrootte is 1. Als in een bin geen (geldig) record is opgeslagen, is de waarde van het sleutelveld in deze bin gelijk aan ∞ .
- C3. Synoniemen zijn door een enkelvoudige ketting met elkaar verbonden. De ketting is ongeordend.
- C4. Als een te verwijderen record in een bin is opgeslagen en de bijbehorende ketting van synoniemen niet leeg is, moet het eerste record van de ketting in de bin worden opgeslagen.
- C5. Voor adresbepaling van een bin wordt gebruik gemaakt van de functieprocedure $\text{conv}(s)$, die aan conv de adreswaarde toekent via sleutelconversie op s .
- C6. Na verwijdering komt eventueel een plaats in de overflow-ruimte vrij. Via de procedure $\text{voeginoverflowvrij}(b, a)$ wordt deze vrije plaats met adres a aan de beschikbare overflowplaatsen van bestand b toegevoegd.

In de mutatieverwerking zijn nu de volgende stappen te onderscheiden:

- mutatierecord inlezen
- het bijbehorende stamrecord opsporen
- indien gevonden
 - muteren
 - eventueel verwijderen met aanpassing van sleutelwaarde in bin òf van recordinhoud van bin òf van kettingstructuur
 - inclusief de gevraagde output op de juiste momenten
- indien niet gevonden, dan foutmelding.

De programmaschets ziet er als volgt uit:


```

begin type record strec(sl,saldo,data,verw),
        uprec(sl,data);
        file stbest of strec
endtype;
var stb: stbest endvar;

ready (stb);

begin var sr,vsr: strec; ur: uprec;
        adres,binadres,vadres: verw;
        gevonden: boolean
endvar;
accept(ur);
adres := conv(ur.sl); binadres := adres;
gevonden := true;
read(stb,adres,sr);
if sr.sl = ∞
then gevonden := false           %"lege"bin%
else while sr.sl ≠ ur.sl and sr.verw ≠ -1
do vadres := adres; vsr := sr;
  adres := sr.verw;
  read(stb,adres,sr)
od;
if sr.sl ≠ ur.sl
then gevonden := false
else print(sr);                  %vóór mutatie%
  sr := sr ⊕ ur;
  print(ur);
  print(sr);                     %ná mutatie%
  if sr.saldo < 0
  then if adres = binadres
  then if sr.verw = -1           %record in bin
                                verwijderen%
  then sr.sl := ∞ %lege ketting%
  else voeginoverflowvrij(stb,sr.verw);
  read(stb,sr.verw,sr);
  fi;
  write(stb,adres,sr)
  else vsr.verw := sr.verw;
  write(stb,vadres,vsr);
  voeginoverflowvrij(stb,adres)
  fi; print("record verwijderd")
  else write(stb,adres,sr)
  fi
fi
fi;
if not gevonden
then print("sleutel niet aanwezig")
fi
end;
save(stb)

end

```

OPGAVEN

- 6.1. a. Wanneer is bij directe organisatie met directe relatie sprake van 100% bezetting van het gegevensbestand?
 b. Geef een programmaschets voor het invoegen van een record in een direct georganiseerd bestand met directe relatie.
- 6.2. Geef een programmaschets voor het zoeken van een record in een direct georganiseerd bestand met tabelrelatie.
- 6.3. Bepaal voor een directe organisatie met funktionele relatie m.b.v. een Poissontabel de gemiddeld te verwachten overflow (als percentage van het totaal aantal records in het bestand) voor de volgende combinaties van bingrootte en vullingsgraad:

<u>bingrootte</u>	<u>vullingsgraad</u>
1	1.0
5	1.0
1	0.7
1	0.75
10	0.8

- 6.4. Deze opgave is bedoeld als 'verdere uitbouw' van het voorbeeld, waarop tabel 1 betrekking heeft. Er wordt dus uitgegaan van de in tabel 1 gegeven records.

Vul het ontbrekende in onderstaande tabel in.

aantal bins	bin- grootte	vullings- graad	% overflow	
			berekend uit gegevens	theoretisch (volgens tabel 3)
100	1	100%	31%	37%
	2	50%	8%	10%
	3			
50	2			
	3			
	4			
25	4			
	5			
	6			
	8			
20	4			
	5			
	6			
	8			
10	10			
	9			
	10			
	12			

- 6.5. Gegeven is een 'bestand' van 12 records. Dit bestand wordt direct georganiseerd met funktionele relatie. Hiervoor zijn beschikbaar 4 primaire sporen (genummerd 0, 1, 2, 3) en 1 overloopspoor. De capaciteit per spoor is 4 records. De 12 records hebben de volgende sleutels: 17, 16, 14, 9, 19, 7, 6, 4, 12, 25, 11, 26 en worden ook in deze volgorde voor opslag aangeboden. Er zijn twee sleutelconversies S_1 en S_2 beschikbaar. Bij S_1 vindt alleen bepaling van spoornummer plaats en wel via het algoritme: $\text{spoornummer} := (\text{sleutel} \bmod 13) \div 4$. De records worden na conversie op de eerstvolgende vrije plaats op het desbetreffende spoor opgeslagen. Is dit spoor vol, dan volgt uiteraard plaatsing in het overflow-spoor. Bij S_2 vindt op gelijke wijze als bij S_1 bepaling van spoornummer plaats, maar bovendien wordt de plaats (0, 1, 2, 3) op het spoor vastgelegd door het algoritme: $(\text{sleutel} \bmod 13) \bmod 4$. Is deze plaats bezet dan wordt het record in de overflow geplaatst. Als bijv. de records met sleutels 22 en 35 als eerste voor opslag zouden worden aangeboden, dan zouden deze records bij gebruik van S_1 op de 0e resp. 1e plaats van spoor 2 worden opgeslagen en bij gebruik van S_2 op de 1e plaats van spoor 2 resp. in de overflow.
- Geef aan op welke plaatsen genoemde 12 records worden opgeslagen bij gebruik van S_1 resp. S_2 .
 - Met welke bingrootte en vullingsgraad wordt in feite bij gebruik van S_1 resp. S_2 gewerkt? Vergelijk het resultaat sub 1 met het theoretisch te verwachten resultaat volgens tabel 3.
- 6.6. Geef een programmaschets voor het zoeken van een record in een direct georganiseerd bestand met funktionele relatie (bingrootte = 1).
- 6.7. Gegeven is een bestand van 10000 records met tabelrelatie. Voor het (relatief) adres zijn dus 4 karakterposities nodig. De recordsleutel bestaat uit 6 karakters. De maximale buffergrootte voor in- en uitvoer bedraagt 1000 karakterposities.
- Beredeneer waarom voor dit geval een hiërarchisch opgebouwde tabel de voorkeur verdient boven een niet-hiërarchische tabel.
 - (Nu verder veronderstellende dat met een hiërarchische tabel wordt gewerkt.)
 - Hoe zijn de hoofdtabellen en de subtabellen opgebouwd?
 - Beschrijf het zoeken van een bepaald record.
 - Beschrijf het invoegen van een nieuw record.
- 6.8. Een bestand bestaat uit 10000 records. Het sleutelveld van elk

record bestaat uit 9 cijfers. Het bestand wordt direct georganiseerd op schijf gezet. Bij deze organisatie wordt gebruik gemaakt van een funktionele relatie. We gaan er vanuit dat de gebruikte randomizer een homogene verdeling van de geconverteerde sleutels geeft.

- a. Zou gebruik van de directe relatie in dit geval zinvol kunnen zijn?
- b. Voor de opslag wordt een schijvenpakket gebruikt met 200 cilinders en 10 sporen per cilinder. Verder wordt gewerkt met een binsize van 5 records. Elk spoor kan 4 bins bevatten. Voor de overflow wordt anderhalf maal de te verwachten overflow gereserveerd.
 1. Hoeveel cilinders moeten voor dit bestand (dus inclusief overflow-ruimte) worden gereserveerd als een vullingsgraad van 80% wordt aangehouden?
 2. Als de te verwachten overflow hoogstens 5% mag bedragen, hoeveel cilinders moeten dan minimaal worden gereserveerd?

- 6.9. Gegeven is een direct georganiseerd werknemersbestand WB met funktionele relatie. In elk werknemersrecord komen o. a. voor het sleutelveld wnr (werknemersnummer) en het veld anr (afdelingsnummer) van de afdeling waartoe de werknemer behoort.

Verder is ook gegeven een direct georganiseerd afdelingenbestand AB met tabelrelatie. Het bijbehorende tabelbestand is sequentieel georganiseerd op oplopende sleutelwaarde. In elk afdelingsrecord komt o. a. het sleutelveld anr (afdelingsnummer) voor.

De bij ieder afdelingsrecord behorende werknemersrecords zijn onderling en met dit afdelingsrecord verbonden met behulp van een dubbele ringstructuur. Elk werknemersrecord is uiteraard in precies één ring opgenomen. Vanuit elk afdelingsrecord wordt verwezen naar een 'dummy' record met sleutelwaarde 0, en een 'dummy' record met sleutelwaarde ∞ . Deze dummy-records hebben dezelfde indeling als een werknemersrecord. Tussen deze twee dummyrecords zijn werknemersrecords in sleutelvolgorde opgenomen. Het aantal 'echte' werknemersrecords kan voor sommige ringen nul zijn.

Geef een programmaschets voor de bestandsverwerking, nodig in verband met de overplaatsing van de werknemer met sleutelwaarde a uit zijn huidige afdeling naar de afdeling met sleutelwaarde b. Er moet rekening worden gehouden met de mogelijkheid dat de nieuwe afdeling gelijk is aan de huidige afdeling. Verder moet er rekening worden gehouden met de eventueel

optredende fouten, dat de gegeven sleutelwaarde a niet in het bestand WB voorkomt en/of de gegeven sleutelwaarde b niet in het bestand AB voorkomt.

Er mag gebruik worden gemaakt van de volgende, niet nader uit te werken, functieprocedures:

- zoek1 (wnr): levert het adres van het record met sleutelwaarde wnr indien dit record in het bestand WB aanwezig is; anders levert deze procedure de waarde -1;
- zoek2 (anr): levert het adres van het record met sleutelwaarde anr, indien dit record in het bestand AB aanwezig is; anders levert deze procedure de waarde -1.

7 COMBINATIES VAN GRONDVORMEN VAN BESTANDSORGANISATIE

In de hoofdstukken drie en zes werden de grondvormen van bestandsorganisatie besproken, nl. de sequentiële en de directe bestandsorganisatie. Wij zullen nu aandacht besteden aan twee mengvormen en wel de zogeheten

- index-sequentiële bestandsorganisatie met gebruikmaking zowel van een tabelstructuur (in 7.1) als van een boomstructuur (in 7.3), en
- index-directe bestandsorganisatie (in 7.2).

7.1 INDEX-SEQUENTIELE BESTANDSORGANISATIE MET TABELLEN

7.1.1 Inleiding

Het zal nogal eens voorkomen dat voor een informatiesysteem zowel sequentiële als directe toegang op een bestand wordt geëist. Men denke bijv. aan een artikelenbestand of een plaatsreserveringsbestand, waarvoor ten behoeve van het samenstellen van (periodieke) overzichten sequentiële toegang en ten behoeve van raadpleging directe toegang nodig is.

Het is dan ook niet verwonderlijk dat er een bestandsorganisatie bestaat, die men zeer toepasselijk direct-sequentieel of sequentieel-direct zou kunnen noemen. In feite is een andere benaming in gebruik en wel die van index-sequentieel.

De eis van zowel sequentiële als directe toegang legt natuurlijk de nodige beperkingen op aan de opslagmogelijkheden van het bestand.

Door de eis van directe toegang zal alleen een adresseerbaar geheugenmedium in aanmerking komen. Dit betekent in feite dat alleen schijfgeheugens (of schijfachtige geheugens) te gebruiken zijn. Dit heeft op zijn beurt weer de nodige consequenties voor de opslagstructuur.

7.1.2 Structuur voor sequentiële en directe toegang

Men kan trachten een bestand direct-sequentieel toegankelijk te maken door uit te gaan van een

- a. directe organisatie en additionele voorzieningen te treffen voor sequentiële toegang, of
- b. sequentiële organisatie en additionale voorzieningen te treffen voor directe toegang.

Ad a.

Bij de directe organisatie maakten we onderscheid tussen de directe relatie, tabelrelatie en funktionele relatie. De directe relatie kunnen we hier buiten beschouwing laten, want dan hebben we in feite ook al te maken met een sequentieel bestand.

Bij de tabelrelatie is sequentiële verwerking in principe mogelijk. Aangezien echter de records in het gegevensbestand niet op volgorde van sleutelwaarde liggen, zal bij sequentiële verwerking geen tijdswinst te boeken zijn omdat de gemiddelde toegangstijd per record (vrijwel) gelijk is aan de toegangstijd bij directe verwerking. Zelfs een additionele voorziening in de vorm van een geordende ketting in het gegevensbestand zet weinig of geen zoden aan de dijk. Immers, in dit geval vervalt weliswaar de (meestal te verwaarlozen) zoektijd in de tabel, maar de gemiddelde toegangstijd tot een record blijft dezelfde.

Bij de funktionele relatie is zonder hulptabellen sequentiële verwerking niet eens mogelijk. Een geordende ketting van records heeft niet alleen dezelfde bezwaren als hierboven, maar geeft grote problemen bij mutaties omdat we, zonder langs de ketting te lopen, eigenlijk niet weten welke sleutelwaarden in het bestand voorkomen.

Samenvattend lijkt het erop dat, afgezien van de in de praktijk weinig bruikbare directe relatie, uitgaande van een directe organisatie geen redelijke voorziening te treffen is voor sequentiële toegang.

Ad b.

Kan het eigenlijk zonder additionele voorzieningen? Immers een sequentieel en consecutief opgeslagen bestand op een adresseerbaar medium voldoet aan alle voorwaarden voor binair zoeken. Dit betekent echter voor bijvoorbeeld een bestand van 10000 records nog altijd zo'n 15 stappen bij binair zoeken, waarbij elke stap een leesopdracht inhoudt! En dan te bedenken dat de directe organisatie gemiddeld maar iets meer dan 1 leesopdracht vergt! Bovendien moet dan bij alle mutaties het bestand consecutief opgeslagen blijven, hetgeen meestal tot een vrij kostbaar onderhoud zal leiden.

Ten behoeve van directe toegang tot een sequentieel bestand zou men in eerste instantie kunnen denken aan een additionele voorziening in de vorm van een (hulp-)tabel met alle paren (sleutelwaarde, record-

adres). Men moet daarbij echter bedenken dat voor een groot bestand deze tabel nog behoorlijke afmetingen kan hebben (en mogelijk vele leesopdrachten kan vergen), en dat bij mutaties op het bestand ook deze grote tabel bijgewerkt moet worden. Is het mogelijk zo'n grote tabel en zijn bewerkingstijd te reduceren? Dit kan gelukkig in verschillende opzichten gebeuren.

In de eerste plaats hebben wij bij genoemde hulptabel nog geen gebruik gemaakt van het feit dat, bij het laden van het bestand, de records consecutief worden opgeslagen. Houden wij hiermee wel rekening dan blijkt dat het in feite niet nodig is het recordadres in de tabel op te nemen. Wij gaan er hierbij, zoals in voorgaande hoofdstukken van uit, dat met het recordadres het relatieve adres wordt bedoeld. Wij veronderstellen uiteraard ook dat de sleutelvolgorde in de tabel gelijk is aan die in het bestand. Opname van het relatieve adres in de tabel is niet nodig omdat de plaats van de sleutelwaarde in de tabel ook tevens het relatieve adres aangeeft van het record met deze sleutelwaarde. Als bijv. in de tabel de sleutelwaarde 10901 op de 73e plaats in de tabel staat, betekent dit dat het record met sleutelwaarde 10901 op relatief adres 73 in het bestand is opgeslagen.

In de tweede plaats kunnen wij gebruik maken van het feit dat de tabel, evenals het bestand, sequentieel geordend is. Daarom kunnen wij ons beperken tot een tabel die slechts een gedeelte van de sleutelwaarden bevat. Als bijv. bij het bovengenoemde bestand van 10000 records een tabel wordt gemaakt met de sleutelwaarden van elk 10e record, dan zal men een tabel hebben met nog maar 1000 ingangen. Zo'n tabel geeft dan van de meeste records de plaats niet meer precies maar alleen bij benadering aan. Deze tabel is in één of hooguit enkele leesopdrachten in te lezen en daarna vergt binair zoeken in de tabel in het werkgeheugen uiteraard een te verwaarlozen tijd. Zo'n tabel wordt dikwijls een *index*tabel genoemd, daarmee aanleiding gevend tot de reeds vermelde (op het eerste gezicht merkwaardige) benaming: *index-sequentiele bestandsorganisatie*. Wij zullen hier echter het woord *traject*tabel gebruiken om daarmee beter tot uitdrukking te brengen dat het om een tabel gaat, waarmee kan worden vastgesteld in welk geheugentraject wij (verder) moeten zoeken. In de *traject*tabel wordt dan van elk traject de sleutelwaarde opgenomen van het record dat aan het begin of aan het einde van het traject is opgeslagen.

In de derde plaats is de bewerkingstijd van een *traject*tabel te reduceren (vooral door het kleinere aantal leesopdrachten) door deze hiërarchisch op te bouwen. Daarvoor is echter een nog wel zo belangrijke reden. Immers zoals wij aan het eind van hoofdstuk 3 al zagen, is het voor een sequentieel bestand op een adresseerbaar medium niet efficiënt bij onderhoud steeds het hele bestand over te schrijven. Dit betekent dat op allerlei plaatsen in het bestand de consecutieve opslag 'lokaal' verstoord kan zijn. Met behulp van hiërarchisch opgebouwde *traject*tabellen kan men goed op deze lokale verstoringen inspelen.

Hiërarchisch opgebouwde trajecttabellen voor het genoemde bestand van 10000 records zouden bijv. kunnen bestaan uit één hoofdtrajecttabel met 50 sleutelwaarden en verder nog 50 trajecttabellen met ieder 20 ingangen. In de hoofdtabel wordt dan de sleutelwaarde van elk 200e record opgenomen en in elk van de trajecttabellen wordt dan van de desbetreffende groep van 200 records de sleutelwaarde van elk 10e record opgenomen.

Opmerking. Teneinde de onderlinge vaste positie van records, waarvan de sleutelwaarden in een trajecttabel voorkomen, eenvoudig te realiseren (en evenzo de onderling vaste posities van trajecttabellen) plaatst(e) men veelal die records (resp. trajecttabellen) aan bepaalde fysieke 'uiteinden', nl. sporen en cilinders. Men sprak daarom over spoortabellen en cilindertabellen i. p. v. over trajecttabellen en hoofdtrajecttabellen. Een record met een bepaalde sleutelwaarde k moet dan te vinden zijn op het spoor aan welks einde het record geplaatst is waarvan de sleutelwaarde de kleinste in de trajecttabel is die groter is dan k . Zoals gezegd willen wij de bestandsstructuur behandelen op het niveau van relatieve adressen. Het heeft dan ook geen zin de begrippen cilindertabel en spoortabel van het lager gelegen fysieke niveau te gebruiken.

Het spreekt vanzelf dat de hiërarchische opbouw van trajecttabellen desgewenst nog verder kan gaan dan twee 'lagen'. Zo zou bijv. voor een sequentieel, strikt consecutief opgeslagen bestand van 100 000 records de opbouw kunnen bestaan uit

1	supertrajecttabel,	S (met 25 ingangen voor ieder 4000e record)
25	hoofdtrajecttabellen,	$H_1 \dots H_{25}$ (ieder met 20 ingangen voor ieder 200e record)
500	trajecttabellen	$T_1 \dots T_{500}$ (ieder met 20 ingangen voor ieder 10e record).

Voor het zoeken van een record met een gegeven sleutelwaarde zsl (zoeksleutel) moet dan eerst de supertrajecttabel worden geraadpleegd om de hoofdtrajecttabel H_i te vinden dank zij het feit dat de i^e ingang in S de kleinste sleutel is die niet kleiner is dan zsl . In de hoofdtrajecttabel bepaalt men dan het nummer j van de ingang waarvoor weer geldt dat deze ingang de kleinste is die niet kleiner is dan zsl . In de trajecttabel met nummer $20(i-1)+j$ bepaalt men tenslotte het nummer van de ingang waarvoor andermaal geldt dat deze ingang de kleinste is die niet kleiner is dan zsl . Daarmee is dan het traject bekend waarin het gezochte record moet liggen (aangenomen dat dit record inderdaad aanwezig is in het bestand).

De opbouw van de trajecttabellen vindt meestal plaats tijdens het (sequentieel) laden van het bestand. Bij een reorganisatie van het

bestand (dus bij opnieuw laden) zullen ook de tabellen opnieuw gemaakt moeten worden.

7.1.3 Consequenties van bestandsmutaties

In het voorgaande hebben wij een algemeen beeld geschetst van de structuur van de index-sequentiële bestandsorganisatie. Dit beeld zou volledig zijn voor een bestand zonder enige invoeging of verwijdering van records. Dit komt echter zelden of nooit voor. Het is dan ook zaak a. en het bestand niet of zelden te hoeven reorganiseren (wegens het vele werk)

- b. en toch de sequentiële en (vooral) directe toegang zo te houden dat deze aan redelijke maatstaven blijft voldoen (bijv. dat voor directe toegang hooguit enkele leesopdrachten nodig zijn).

Het zal dikwijls niet eenvoudig zijn aan punt b te voldoen. Immers men staat voor het dilemma van enerzijds redelijke toegang ten koste van relatief kostbare onderhoudsvoorzieningen, anderzijds relatief goedkoop onderhoud ten koste van minder goede toegankelijkheid. Het is dan ook niet te verwonderen dat er allerlei varianten van de index-sequentiële bestandsorganisatie bedacht zijn en gebruikt worden om een redelijke (we spreken maar liever niet van optimale) oplossing voor dit dilemma te verkrijgen.

7.1.4 Enkele varianten van de I-S-organisatie

Wij zullen nu eerst enkele kenmerken bespreken die bij (vrijwel) alle varianten zijn terug te vinden. Verder zullen wij drie varianten apart bespreken en deze ook toelichten aan de hand van eenzelfde voorbeeld.

Bij de behandeling van de varianten wordt uiteraard het geval buiten beschouwing gelaten van een volkomen stabiel bestand (zonder enig verloop of groei). Wij veronderstellen steeds een tabelopbouw in twee lagen: hoofdtrajecttabel en trajecttabellen. Bij elke variant zal nodig zijn:

- een overloopvoorziening,
- vastlegging van informatie in de trajecttabellen betreffende de (kettingen van) records in de overloop.

Betreffende de overloopvoorziening veronderstellen we voor ieder hoofdtraject een overloopgebied (hoofdtrajectoverloop) en een voor het gehele bestand beschikbaar overloopgebied (bestands-overloop). Dit laatste kan nodig zijn als een hoofdtrajectoverloop vol is, hetgeen wij in de hierna volgende tekst echter niet zullen meemaken. Desgewenst kan men deze veronderstellingen nog uitbreiden met een overloopgebied voor ieder traject doch dit brengt geen nieuwe facetten met zich mee.

VARIANT A

- A1. Alle hoofdtrajecten worden op dezelfde wijze verdeeld in een primair gebied en een overloopgebied. Als bijv. elk hoofdtraject 20 trajecten bevat, zou het primaire gebied kunnen bestaan uit de eerste 17 trajecten en het overloopgebied uit de laatste 3 trajecten.
- A2. Bij het (hernieuwd) laden van het bestand wordt elk traject van het primaire gebied, behalve mogelijk het laatste, geheel gevuld met records.
- A3. Invoeging van een nieuw record gebeurt door plaatsing van dit record in de overloop van het passende hoofdtraject en vastlegging in een overloopketting.
- A4. De toegevoegde records, die door hun sleutelwaarden bij een bepaald traject horen en in de overloop staan, zijn door een enkelvoudige sequentiële ketting met elkaar verbonden. Het beginadres van elke ketting (-1 voor de lege ketting) is opgenomen in de bijbehorende trajecttabel.
- A5. Verwijdering van een record vindt als volgt plaats:
 - van een record in het primaire gebied door een bepaald veld (flag) in dit record een waarde te geven, waaruit blijkt dat het record als niet aanwezig moet worden beschouwd;
 - van een record in een overloopketting door het record uit deze ketting te verwijderen en de recordpositie vrij te geven voor een nieuw record.
- A6. Bij het (hernieuwd) laden van het bestand worden in de hoofdtrajecttabel en in elke trajecttabel de hoogste sleutelwaarde voor ieder hoofdtraject c. q. traject opgenomen. Deze sleutelwaarden in de tabellen kunnen alleen bij reorganisatie, dus bij hernieuwd laden, veranderen. Zelfs als een record wordt verwijderd waarvan de sleutelwaarde in een tabel voorkomt, dan nog wordt deze sleutelwaarde in de desbetreffende tabel niet door een ander vervangen. Op het eerste gezicht lijkt deze opzet niet flexibel. Het zal echter blijken, met name bij de programmaschetsen, dat deze aanpak onnodige verzwarende van de programmatuur voorkomt.

VARIANT B

- B1. Als A1.
- B2. Bij het (hernieuwd) laden van het bestand wordt elk traject van het primaire gebied slechts voor een constant gedeelte gevuld met records.

- B3. Invoeging van een nieuw record gebeurt door plaatsing op de goede plaats in een primair traject of in de bij dit traject behorende ketting, echter met dien verstande dat de sleutelwaarden van records in elk primair traject kleiner zijn dan die van de bijbehorende overlooprecords.
- B4. Als A4.
- B5. Verwijdering van een record vindt als volgt plaats:
- in een primair traject door opschuiven van records met hogere sleutelwaarden en opnemen van een eventueel eerste ketting-record in het primaire traject;
 - in een ketting door verwijdering uit die ketting en de recordpositie vrij te geven voor een nieuw record.
- B6. Als A6. Bovendien is per traject het totale aantal records in het primaire gebied en het overloopgebied tezamen, in de trajecttabel opgenomen.

VARIANT C

- C1. Als A1.
- C2. Als A2.
- C3. Invoeging van een nieuw record gebeurt door plaatsing in de overloop en vastlegging in de enkelvoudige sequentiële ketting van een primair traject.
- C4. Alle records die door hun sleutelwaarden tot een primair traject horen zijn opgenomen in een enkelvoudige sequentiële ketting. Ook de records in het primaire traject zelf zijn in deze ketting opgenomen. Het beginadres van elke ketting is opgenomen in de bijbehorende trajecttabel. Een ketting kan leeg zijn als alle records in het primaire gebied verwijderd zijn en er ook geen bij dit primaire traject behorende overlooprecords (meer) zijn. In zo'n geval van de lege ketting zal genoemd beginadres de waarde -1 hebben.
- C5. Verwijdering van een record geschiedt door verwijdering uit bovengenoemde ketting en in geval van een overlooprecord de recordpositie vrij te geven voor een nieuw record.
- C6. Als A6.

Wij zullen nu aan de hand van een getallenvoorbeeld bovengenoemde drie varianten toelichten.

VOORBEELD

Gegeven is dat

- elk hoofdtraject tien trajecten bevat,
- elk traject, in de varianten A en C, vijf recordposities bevat en in de variant B zes recordposities,
- per hoofdtraject de eerste 8 trajecten tot het primaire gebied behoren en de laatste 2 tot het overloopgebied,
- de hoofdtrajecttabel en de trajecttabellen in een apart (hulp)bestand zijn opgeslagen,
- de administratie van vrije plaatsen in het overloopgebied wordt bijgehouden en wel zodanig dat vrijgekomen plaatsen onderaan aan de lijst van vrije plaatsen worden toegevoegd. Bij de bestandsopbouw bestaat de lijst van vrije plaatsen uit de 1e positie in traject 9 tot en met de 5e positie (6e positie in variant B) in traject 10,
- de inhoud van het 3e hoofdtraject bij het laden van het bestand is als hieronder (van elk record wordt alleen de sleutelwaarde gegeven).

traject	hoofdtraject 3 (bij het laden van het bestand)				
1	305	361	382	399	407
2	415	427	436	440	442
3	451	457	462	463	465
4	469	471	473	476	477
5	485	492	497	506	511
6	512	513	526	528	533
7	547	549	561	567	569
8	580	585	592	603	607
9	overloop				
10					

Verder is gegeven dat in de loop van de tijd de volgende mutaties (in deze volgorde en ieder apart) moeten worden uitgevoerd:

sleutelwaarde mutatie (I = invoegen; V = verwijderen)

539	I
464	I
457	V
529	I
428	I
465	V
465	I
458	I
453	I
520	I
526	V
528	V

465	V
512	V
459	I
428	V

Na verwerking van deze mutaties zal de inhoud van hoofdtraject 3 als volgt zijn voor de varianten A en C:

traject					
1	305	361	382	399	407
2	415	427	436	440	442
3	451	-	462	463	-
4	469	471	473	476	477
5	485	492	497	506	511
6	-	513	-	-	533
7	547	549	561	567	569
8	580	585	592	603	607
9	539	464	529		
10	458	453	520	459	

met de overloopkettingen:

voor A: bij traject 3: 453-458-459-464
 bij traject 6: 520-529
 bij traject 7: 539

en enkele trajectkettingen:

voor C: bij traject 3: 451-453-458-459-462-463-464
 bij traject 6: 513-520-529-533
 bij traject 7: 539-547-549-561-567-569

en voor de variant B

traject					
1	305	361	382	399	407
2	415	427	436	440	442
3	451	453	458	459	462 463
4	469	471	473	476	477
5	485	492	497	506	511
6	513	520	529	533	
7	539	547	549	561	567 569
8	580	585	592	603	607
9			464		
10					

Na uitvoering van de genoemde mutaties staat in de trajecttabel bij hoofdtraject 3 o. a.

variant A		variant B		variant C
traject	beginadres ketting	begin ketting	aantal per traject	beginadres ketting
1	-1	-1	5	1, 1
2	-1	-1	5	2, 1
3	10, 2	9, 3	7	3, 1
4	-1	-1	5	4, 1
5	-1	-1	5	5, 1
6	10, 3	-1	4	6, 2
7	9, 1	-1	6	9, 1
8	-1	-1	5	8, 1

7.1.5 Programmaschetsen voor direct zoeken in een I-S-bestand

Wij zullen nu voor elk der drie varianten A, B en C uit de vorige paragraaf een programmaschets geven voor het opzoeken en afdrukken van een record met een gegeven sleutelwaarde. Hierbij dient rekening gehouden te worden met de eventueel optredende fout dat de opgegeven sleutelwaarde niet in het bestand aanwezig is.

Wij zullen daarbij van de volgende veronderstelling uitgaan:

- voor het bestand zijn 40 hoofdtrajecten gereserveerd,
- elk hoofdtraject bestaat uit 40 trajecten,
- elk traject bevat 10 recordposities,
- per hoofdtraject behoren de eerste 35 trajecten tot het primaire gebied en de laatste 5 trajecten tot het overloopgebied,
- de hoofdtrajecttabel en de trajecttabellen zijn in een apart (hulp)-bestand opgeslagen.

Eenvoudigheidshalve zullen wij voor de drie varianten uitgaan van eenzelfde structuur voor het stamrecord en eenzelfde structuur voor het trajecttabelrecord. Dit betekent dat in elk stamrecord het veld verw (voor adresverwijzing) voorkomt en het boolean veld del voor het bijhouden of een record wel of niet verwijderd is, en het trajecttabelrecord een array is met de structuur `a[1:40]` of element, waarbij element de volgende recordstructuur heeft: `element(sl, verw, aantal)`, met de betekenis:

`element.sl` : sleutelwaarde;
`element.verw` : beginadres ketting;
`element.aantal` : aantal records (alleen voor variant B).

Zo komen wij tot een gelijkkluidend beginstuk voor de drie varianten en wel:

```

begin type record strec(sl,data,del,verw),
                                element(sl,verw,aantal);
                                trajtabrec[1:40] of element;
                                file stbest of strec,
                                trajtabbest of trajtabrec
endtype;
var stb: stbest; trajb: trajtabbest
endvar;

ready(stb,trajb)

```

Ook de initialisatie van het programma zal voor alle drie varianten dezelfde zijn, immers in alle drie gevallen zal eerst het traject bepaald moeten worden, waar het gezochte record zich moet bevinden. Voor deze initialisatie maken wij gebruik van de volgende variabelen:

zsl : (zoek)sleutel van gevraagd record
htrjt resp. trjt: array voor het inlezen van de hoofdtrajecttabel resp. een trajecttabel

ih en it : wijzers in htrjt en trjt
en de procedures hzoek en tzoek, geheel analoog aan die in paragraaf 6.6 (behandeling met tabelrelatie). Dit gemeenschappelijke stuk initialisatie zal er dan als volgt uitzien:

```

accept(zsl);
read(trajb, 0, htrjt);
ih := hzoek(htrjt, zsl);
read(trajb, ih, trjt);
it := tzoek(trjt, zsl);

```

De verdere verwerking zal nu voor iedere variant behoorlijk verschillen.

VARIANT A

In deze variant moet na bovenstaande initialisatie eerst in het primaire traject worden gezocht naar het gevraagde record, en als dat geen resultaat oplevert, moet het zoekproces eventueel worden voortgezet in het overloopgebied. Deze volgorde (zoeken in primair gebied, zoeken in overloopgebied) kan ook worden omgekeerd, aangezien deze twee gebieden onderling niet sequentieel geordend zijn. Bij het zoeken in het primaire gebied moet ook rekening worden gehouden met verwijderde records.

Voor het programma zullen wij gebruik maken van o. a. de volgende variabelen:

gevonden : boolean om aan te geven of record gevonden is
klaar : boolean om aan te geven of men klaar is met het zoeken in het primaire traject.

Wij zullen afspreken dat van een verwijderd record op geen enkele wijze, voor vergelijken van sleutels e.d., gebruik mag worden gemaakt. Houdt men zich niet aan dergelijke (voor de hand liggende) afspraken dan kan dit zeer moeilijk te vinden fouten tot gevolg hebben.

Het (specifieke gedeelte van het) programma voor variant A zal er nu als volgt uitzien:

```

begin var sr: strec; htrjt,trjt: trajtabrec;
        gevonden,klaar: boolean;
        ih,it: integer; zsl: sl; adres: verw
endvar;
initialisatie;                                %zie boven%
adres := (ih-1) * 400 + (it-1) * 10;
gevonden := false; klaar := false; k := 0;
while not klaar
    do k := k+1; read(stb,adres+k,sr);
        if not sr.del
            then gevonden := sr.sl = zsl;
                 klaar := sr.sl ≥ zsl or k = 10
            else klaar := k = 10
        fi
    od;
if not gevonden
    then adres := trjt[it].verw;
        if adres ≠ -1
            then read(stb,adres,sr);
                while sr.sl < zsl and sr.verw ≠ -1
                    do read(stb,sr.verw,sr) od;
                    gevonden := sr.sl = zsl
                fi
            fi;
        if gevonden
            then print(sr)
            else print("niet gevonden")
        fi
end

```

VARIANT B

Hier kunnen wij gebruik maken van het feit dat primair gebied en overloopgebied onderling sequentieel geordend zijn. Aan de hand van de opgegeven sleutelwaarde zsl zal worden bepaald of wij in het primaire gebied of in het overloopgebied moeten zoeken. In deze variant zullen wij dan ook de boolean variabele pr gebruiken om aan te geven of er wel of niet in het primaire gebied moet worden gezocht.

Het (specifieke gedeelte van het) programma voor variant B is nu als onderstaand:

```

begin var sr:strec; htrjt,trjt: trajtabrec;
      klaar,pr,gevonden: boolean;
      h,ih,it,a: integer; zsl: sl; adres: verw
endvar;
initialisatie;                                %zie boven%
gevonden := false;
a: trjt[it].aantal;                               %a: aantal records%
if a > 0
  then if a > 10
    then read(stb,trjt[it].verw,sr);
        pr := sr.sl > zsl
    else pr := true
  fi;
  if pr
    then adres := (ih-1) * 400 + (it-1) * 10;
        klaar := false; k := 0; h := minimum(10,a);
        while not klaar
          do k := k+1;
              read(stb,adres + k,sr);
              klaar := sr.sl > zsl or k = h
          od
        else while sr.sl < zsl and sr.verw ≠ -1
          do read(stb,sr.verw,sr)
          od
        fi;
        gevonden := sr.sl = zsl
  fi;
if gevonden
  then print(sr)
  else print("niet gevonden")
fi
end

```

VARIANT C

Hier kunnen wij gebruik maken van het feit dat voor ieder primair traject een ketting bestaat die alle records van dit traject plus die van het bijbehorende overloopgebied bevat.

Het (specifieke gedeelte van het) programma voor variant C is als volgt:


```

begin var sr: strec; htrjt, trjt: trajtabrec;
      gevonden: boolean;
      ih, it: integer; zsl: sl; adres: verw
endvar;
initialisatie;                                     %zie boven%
gevonden := false;
adres := trjt[it].verw;
if adres  $\neq$  -1
  then read(stb, adres, sr);
      while sr.sl < zsl and sr.verw  $\neq$  -1
        do read(stb, sr.verw, sr)
        od;
        gevonden := sr.sl = zsl
  fi;
  if gevonden
    then print(sr)
    else print("niet gevonden")
  fi
end

```

Men moet erg voorzichtig zijn om aan de gegeven programmaschetsen bepaalde (vergaande) conclusies te verbinden. Immers van het dilemma, waarvan in 7.1.3 sprake was, is in de programmaschetsen slechts een aspect aan bod gekomen, nl. dat van de (wel of niet redelijke) toegang. Voor het onderhoudsaspect zij men verwezen naar (eigen werk naar aanleiding van) de opgaven. Wij zullen deze hoofdpargraaf over index-sequentiële bestandsorganisatie afsluiten met enkele algemene opmerkingen over het onderhoud. Overigens kan het 'narekenen' van het voorbeeld in paragraaf 7.1.4 ook al enig idee over (de problemen van) het onderhoud geven.

7.1.6 Onderhoud van een I-S-bestand

Bij een index-sequentieel bestand dient men steeds in het oog te houden dat dit bestand zowel sequentieel als direct toegankelijk is en dat onderhoud dus ook in een sequentiële verwerkingsgang of per record apart in directe verwerking kan plaatsvinden. Terwijl een sequentieel bestand alleen geschikt is voor hoge file-activity, kan een index-sequentieel bestand dus zowel bij hoge als lage file-activity worden gebruikt. Het is dan ook niet per sé nodig dat de transactierecords worden gesorteerd zoals bij een transactiebestand bij een sequentieel bestand. Sortering wordt echter des te meer gewenst naarmate

- het meer voorkomt dat op een bestaand record meerdere mutaties moeten worden uitgevoerd (afhankelijk van de aard van deze mutaties zal sortering zelfs noodzakelijk kunnen zijn),
- de file-activity groter wordt en er dus meer reden is om tot sequentiële verwerking over te gaan.

Over de drie soorten mutaties (veranderen, invoegen, verwijderen) merken we verder nog op:

Het veranderen van records in een index-sequentieel bestand is, behoudens een soms ingewikkeld zoekproces, een eenvoudige zaak: ook bij directe verwerking is een record vrijwel direct in te lezen, te veranderen en weer op het oude adres terug te schrijven. In het bovenstaande werd reeds gewezen op de wenselijkheid (c.q. noodzaak) van sortering bij meerdere mutaties op eenzelfde record.

Zoals reeds besproken bij de sequentiële organisatie is oppassen geboden ten aanzien van vermeerdering of vermindering van velden, wanneer er een kans is op tussentijds afbreken van het mutatieproces. Hetzelfde geldt voor eventuele controletotalen.

Het toevoegen van records in een index-sequentieel bestand is daarentegen meestal een niet eenvoudige bewerking, vanwege het eventueel

- verplaatsen van records (in het primaire gebied) ter handhaving van de juiste volgorde (zie variant B in 7.1.4),
- veranderen van (gedeelten van) trajecttabellen,
- wijzigen van adresverwijzingen (zie vooral variant C in 7.1.4).

Ook het verwijderen van records kan een vrij ingewikkelde verwerking tot gevolg hebben, met name het verwijderen van records in het primaire gebied. Voor dit laatste geval kiest men daarom dikwijls voor de eenvoudigste oplossing, nl. dat het te verwijderen record als zodanig wordt 'gemerkt', doch verder (fysiek) aanwezig blijft (zie variant A in 7.1.4). Door dit merken wordt dan 'de toegang tot het record afgesloten'.

Uiteraard geldt voor toevoegen en verwijderen hetzelfde als voor veranderen dat het (voorafgaande) zoekproces ook nog vrij ingewikkeld kan zijn.

Wanneer na enige tijd grote aantallen records in overloopgebieden terecht zijn gekomen, en/of overbodig geworden records flink in aantal zijn toegenomen, dan wel een overloopgebied is volgeraakt, moet men met het oog op toegenomen verwerkingstijden overgaan tot een bestandsreorganisatie. Dit gebeurt door het totale bestand in sleutelvolgorde te kopiëren op een nieuwe schijveneenheid, en daarbij overbodige records te verwijderen en overloopgebieden leeg te maken. Bij deze bewerking zal veelal tevens een kopie van het bestand op magneetband vastgelegd worden ten behoeve van bestandsbeveiliging.

7.2 DIRECTE BESTANDSORGANISATIE VOOR EEN BESTAND MET GROTE RECORDLENGTE

In het hoofdstuk over directe bestandsorganisatie hebben wij gezien dat met name bij de funktionele relatie het te verwachten percentage overlooprecords afhankelijk is van de bingrootte en de vullingsgraad. Bij een 'randomizer', die een homogene verdeling van de geconverteerde sleutels geeft, kan de grootte van het te verwachten percentage worden bepaald als bingrootte en vullingsgraad bekend zijn (zie tabel 3 in hoofdstuk 6). Daarbij blijkt dat bij kleine bingrootte en niet al te lage vullingsgraad een hoog percentage overflow is te verwachten, bijv. bij bingrootte 1 en vullingsgraad 80% een overloop van ruim 30% en bij vullingsgraad 50% nog altijd een overloop van ruim 20%. Deze hoge overlooperpercentages kunnen de efficiëntie van de directe toegang op het bestand danig geweld aandoen. Toch zal hier, zonder verdere voorzieningen, niet aan ontkomen kunnen worden bij grote recordlengte, bijv. van enkele duizenden karakters. Dan zal het nl. weinig zin hebben een grote waarde voor de bingrootte te kiezen, daar de bin dan te groot wordt om in zijn geheel behandeld (gelezen) te kunnen worden. Willen wij dan toch gebruik blijven maken van de funktionele relatie, en tegelijkertijd de overflow sterk verminderen, dan kan dit door middel van een additionele voorziening in de vorm van tabellen. Deze tabellen worden dikwijls (vergelijk paragraaf 7.1) index-tabellen genoemd. Daarvandaan de benaming index-directe bestandsorganisatie. Evenals bij de benaming index-sequentieel moeten wij ook hier opmerken dat uit de naam zelf niet duidelijk is op te maken om welke structuur het gaat. Wij zullen nu de opzet van de index-directe organisatie uiteenzetten.

Voor de records zelf wordt een ruimte gereserveerd, zoals voor het gegevensbestand bij de tabelrelatie. Verder wordt ruimte gereserveerd voor tabellen. Elke tabel kan een maximaal aantal paren (sleutelwaarde, adres) bevatten. Tot zover is het eigenlijk hetzelfde als de aanpak bij de (hiërarchische) tabelrelatie. De wijze waarop een tabel wordt geraadpleegd is nu echter verschillend. Uit de sleutelwaarde (van een gevraagd record) wordt nl. via een funktionele relatie, dus via een randomizer, een (relatief) adres afgeleid. Dit adres is dan niet het adres van een record (in het gegevensbestand), maar van een der genoemde tabellen. Elk van deze tabellen is geordend naar (oplopende) sleutelwaarde. Van elk record in het gegevensbestand is in precies één van de tabellen de sleutelwaarde en zijn adres in het gegevensbestand opgenomen. In welke tabel dit is opgenomen hangt uiteraard af van de funktionele relatie. Evenals bij de tabelrelatie geldt ook hier dat een tabel vele paren (sleutelwaarde, adres) zal bevatten. Maar dit betekent in feite dat hier nu sprake is van een grote bingrootte. Als we nu verder nog elke tabel 'ruim bemeten', dus als wij werken met een matige vullingsgraad van de tabellen, is overloop praktisch uitgesloten. Voor dit uitsluiten van overloop moeten

wij dan wel een prijs betalen in de vorm van een extra tabellenbestand, maar dit nadeel zal, bij grote recordlengte, ruimschoots worden vergoed door een meer directe toegang en/of besparing op geheugenruimte.

Ter toelichting zullen wij de index-directe organisatie toepassen op het voorbeeld van tabel 1 van hoofdstuk 6. Wij veronderstellen daarbij het volgende:

- er is ruimte voor 5 tabellen beschikbaar
- elke tabel kan maximaal 40 paren (sleutelwaarde, adres) bevatten
- de records zijn opgeslagen in het gegevensbestand in de volgorde van tabel 1 van hoofdstuk 6. Dus record met sleutelwaarde 710 op adres 1, record met sleutelwaarde 354 op adres 26, enz.
- de sleutelwaarde sl van een record is opgenomen in de tabel waarvan het relatieve adres in het tabelbestand gelijk is aan $sl \bmod 5$
- elke tabel wordt afgesloten met een sleutelwaarde ∞ .

Uit bovenstaande specificaties volgt dat de 3e tabel, dus die op relatief adres 2 in het tabelbestand, de volgende inhoud zal hebben:

sleutelwaarde	adres
057	34
077	66
167	50
172	89
197	88
267	86
277	35
282	10
297	58
327	44
482	57
552	29
562	81
607	21
632	36
707	13
882	94
972	25
987	18
∞	--

Bij geen van de vijf tabellen zal in dit voorbeeld (ook maar in de verste verte) sprake zijn van overloop (zie ook de opgaven). Dit is trouwens niet te verwachten bij een bingrootte van 40 (ofwel 39, als men de laatste plaats reserveert voor de sleutelwaarde ∞) en een vullingsgraad van 50% (ga dit na!).

Met de index-directe organisatie wordt een record dus op een indirecte manier, via een tabel, benaderd. Dit betekent dan ook een extra toegang, nl. op de desbetreffende tabel. Het kan echter vele toegangen (op overloopkettingen) besparen. Bovendien zal het zeker een besparing betekenen op de te reserveren geheugenruimte. Tenslotte zij er nogmaals op gewezen dat in deze paragraaf wordt uitgegaan van een grote recordlengte.

7.3 EEN ANDERE TUSSENVORM VAN DE SEQUENTIELE EN DIRECTE ORGANISATIE MET BEHULP VAN BOMEN

7.3.1 Inleiding

Naast de in 7.1 behandelde methode om een bestand zowel sequentieel als direct te kunnen benaderen is de laatste jaren een andere methode geïntroduceerd. Die maakt gebruik van boomstructuren in plaats van tabellen of misschien beter gezegd de relevante delen van tabellen zijn a. h. w. opgeknipt en op een slimme manier tussen de in een boomstructuur geplaatst bestandsrecords geplaatst. Omdat men bij deze nieuwe methode geen last heeft van bestandsreorganisatie zal hij voor problemen met bepaalde 'afmetingen' ruimere ingang vinden. Los van deze toepassing zal eerst het B-boom concept besproken worden.

Een B-boom is een boom met de volgende eigenschappen:

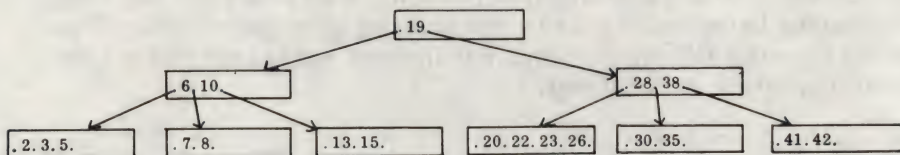
- iedere knoop bevat o. a. tenminste k en ten hoogste $2k$ (sequentieel geordende en consecutief opgeslagen) records, uitgezonderd de wortel van de boom die ook minder dan k records mag bevatten. De grootte k noemt men de orde van de B-boom;
- noemen we het werkelijk aanwezige aantal records in een knoop n , dan wordt geëist dat een knoop 0 of $(n+1)$ opvolgers heeft (verder zonen genoemd; een voorganger wordt ook vader genoemd);
- knopen worden op een bepaalde manier met records gevuld;
- alle knopen zonder zonen (verderop bladeren te noemen) liggen even ver van de wortel, m. a. w. hebben een even lang pad tot de wortel of nog anders gezegd: hebben dezelfde hoogte.

Afhankelijk van de recordgrootte kiest men k bij voorkeur zo groot dat een knoop een spoor van een schijf- of trommelgeheugen vult. Het lezen of schrijven van een spoor in zijn geheel vergt namelijk nauwelijks meer tijd dan dat van een deel van een spoor.

7.3.2 Zoeken van een record

Om de gedachten te bepalen is hierbij een B-boom van de orde 2 met drie niveaus getekend. In iedere knoop zijn slechts sleutelwaarden

uitgeschreven, terwijl met puntjes verwijzingen naar andere knopen zijn aangegeven (in de bladeren 'nul-verwijzingen').



Door de manier waarop een knoop gevuld is kan men lineair (voor kleine k) of binair (als k groter is dan ongeveer 15) een record met een gegeven sleutel zoeken. Als men zo'n record niet kan vinden in een knoop dan

- is het record niet in het bestand aanwezig als de (laatst bekeken) knoop geen zonen heeft,
- moet men anders verder zoeken in de juiste zoon-knoop (het zoeken begint dus in de wortel).

Wat de juiste knoop is wordt aangegeven door een (adres)verwijzing die geplaatst is òf voor het eerste record met een grotere sleutel dan de zoeksleutel òf na het laatste record in het spoor als dit laatste record een kleinere sleutel had dan de zoeksleutel. Ieder record is dus 'omsloten' door 2 verwijzingen, hetgeen verklaart dat een spoor met n records precies $n+1$ (of nul) zonen heeft. Met bijv. 100 records in een spoor hoeft men voor een bestand met 10^6 records slechts 3 sporen te raadplegen! (Dit aantal van 100 records in een spoor stelt in de praktijk natuurlijk wel een bovengrens aan de recordgrootte, en daardoor aan de bruikbaarheid van dit systeem.)

7.3.3 Toevoegen van een record

Voor het eenvoudige zoekproces, dat hiervoor beschreven is, moet de B-boom dus op een bepaalde manier gevuld worden, reeds beginnend bij het eerste record. De werkwijze is daarbij de volgende:

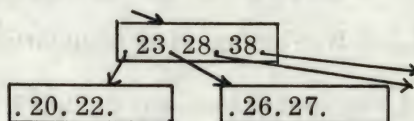
- zoek eerst het blad, waar het in te voegen record gezien zijn sleutel in zou moeten;
- is het blad nog niet vol, plaats dan het nieuwe record op de juiste plaats in het blad (hetgeen enige verschuivingen van andere records kan betekenen);
- is het blad al vol, dan wordt het nieuwe record plus de aanwezige oude records geherdistribueerd over het oude blad en een nieuw door het systeem te leveren blad. Van de herdistributie wordt echter het 'middelste' record uitgesloten, omdat dit teruggeschoven wordt (waarom?) naar de vader van het oude blad en op de goede

plaats in de vaderknoop geplaatst moet worden.

Als bij het terugschuiven van een record de vaderknoop al vol was, wordt op dezelfde wijze de vaderknoop 'gesplitst' en een record verhuist naar de grootvaderknoop. Hoewel niet waarschijnlijk (als de getallen k en $2k$ in de praktijk ver uit elkaar liggen), zou dit terugschuifproces zelfs kunnen resulteren in een opsplitsing van de wortel en het merkwaardige verschijnsel dat een boom aan de wortelkant groeit.

Voegt men aan de eerder getekende boom een record met sleutelwaarde 27 toe, dan resulteert dit in de hiernaast getekende deelboom.

Het is ook duidelijk dat er na een splitsing weer veel ruimte ontstaat in de nieuwe bladeren, zodat er heel wat toevoegingen mogelijk zijn voordat een nieuwe splitsing nodig is.



Voor het toevoegen van een record is dus nodig of gewenst dat

- het systeem vrije knopen (met hun adres) kan uitdelen,
- in iedere knoop ook opgenomen worden: het aantal records in de knoop en een verwijzing naar de vaderknoop (waarom?),
- in het werkgeheugen bufferruimte voor 2 knopen is.

7.3.4 Verwijderen van een record

De eenvoudigste manier om een record te verwijderen is weer het desbetreffende record van een verwijderingskenmerk te voorzien. Als echter in de loop der tijd veel records op die manier 'gemerkt' zijn, gaat dit natuurlijk ten koste van de ruimte die het bestand in zijn geheel in beslag neemt. Zeker bij een groot bestandsverloop is het daarom nodig een record R inderdaad fysiek uit een knoop K te verwijderen, hetgeen een vrij gecompliceerd proces kan zijn.

Als R in een blad met tenminste $k+1$ records zit, hoeven de overgebleven records slechts op te schuiven. Zit R echter in een knoop, die niet een blad is, dan moet R vervangen worden door het meest rechtse record in de linkerzoon (waarom?) van R , uit welke zoon dus een record verdwijnt. Dit proces zal zich 'naar beneden' voortplanten totdat uiteindelijk een record uit een blad 'naar boven' verhuist. Helaas kan zich in het blad B de complicatie voordoen dat het aantal records onder k zakt. Om dat weer goed te maken moet men eerst proberen een record over te nemen uit een linker- of rechterbuur van B . Als dat kan omdat B en buur samen nog meer dan $2k$ records hebben, hoeft tenslotte nog maar één verwisseling met een record uit de gemeenschappelijke vaderknoop plaats te vinden om de goede totaalstructuur te krijgen (waarom?). Kan het echter niet omdat de buurbladeren ook maar k records hadden, dan zit er niets anders op dan blad B te verenigen met een van zijn burens en in het overgebleven blad ook het goede record uit de gemeenschappelijke

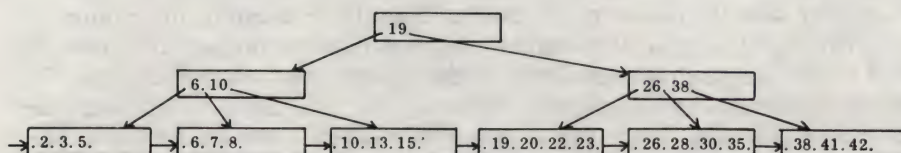
vaderknoop te plaatsen (waarom moet daar een verwijzing uit verdwijnen?). Omdat de vaderknoop nu 1 record minder heeft kan dit proces zich 'naar boven' voortplanten en kan in principe de boom uiteindelijk 1 lager worden. Dit maximum aan 'pech' zal zich echter in de praktijk door de kleine kans dat alle knopen zo kritisch gevuld zijn niet gauw voordoen.

7.3.5 B^+ -bomen voor sequentiële en directe toegang

Na het voorgaande zal duidelijk zijn waarom B-bomen in principe geschikt zijn voor directe toegang tot een bestand met n records. Men hoeft slechts wat meer dan $k \log n$ (waarom?) sporen in te lezen en daarin binair te zoeken om een record te vinden, toe te voegen of te verwijderen, (omdat in de praktijk k groot moet zijn, is de boom zo 'plat' dat men beter over bodembedekker kan spreken). Ook voor sequentiële verwerking is een goed platte B-boom bruikbaar, omdat telkens nadat hooguit $k \log n$ sporen gelezen zijn, aan de verwerking van tussen k en $2k$ records begonnen kan worden.

Toch zal men tegen de besproken B-boom het bezwaar kunnen opwerpen dat het in de praktijk moeilijk is platte bomen te maken en dat bij toevoegingen of verwijderingen van hele records geschoven moet worden in een spoor. Bij de variant van de wel zo genoemde B^+ -bomen gebruikt men de boom dan ook alleen om in ieder spoor slechts de zoekinformatie op te slaan, bestaande uit paren (recordsleutel, wijzer naar het bij de sleutel horende record). Een voordeel van deze methode is dat het nu onafhankelijk van de recordgrootte eenvoudiger is grote waarden van k te kiezen, hetgeen essentieel is voor de efficiëntie! De records zouden bijvoorbeeld eenvoudig in volgorde van aanbieding op vrije sporen geplaatst kunnen worden, waarbij vrije plaatsen in een ketting zijn opgenomen. Terugdenkend aan de tabelrelatie bij de directe toegangsmethoden ziet men dat de B^+ -boom eigenlijk ook als een soort tabelrelatie op te vatten is, waarbij tabelelementen echter in een boom geplaatst zijn.

Een andere verfijning die wel wordt aangebracht is dat in 'opeenvolgende' bladeren alle sleutels – dus ook die uit 'hogere' knopen – staan. Als deze bladeren dan ook nog naar elkaar wijzen komt dit de sequentiële verwerking wederom ten goede (waarom?). Een met de



eerder getekende B-boom corresponderende B^+ -boom is hierboven getekend. Het toevoegen en verwijderen van records is bij B^+ -bomen gecompliceerder dan bij B-bomen. Wij laten het echter bij deze vluchtige introductie tot B-bomen (zie voor meer details het overzichtsartikel van D. Comer.

OPGAVEN

- 7.1 In de beschrijving van een index-sequentieel georganiseerd bestand, dat wordt gebruikt in een bepaald informatiesysteem, komen de volgende zinsneden voor:
- '... Bij aanbidding van een nieuw record wordt dit record tijdelijk in de eerste vrije plaats in de overflow opgeslagen; bij de eerstvolgende reorganisatierun wordt het dan op zijn logische plaats gezet. Deze reorganisatieruns blijken veelvuldig plaats te moeten vinden...'
- ... Bij de wekelijkse verwerking van programma P15 is het zinvol en uit efficiency-overwegingen eigenlijk noodzakelijk de mutaties eerst op sleutelvolgorde te sorteren...'
- Wat moet door het systeem bij invoeging van een nieuw record worden gedaan om het bestand sequentieel toegankelijk te houden?
 - Wat zou worden bedoeld met een reorganisatierun?
 - Wat zou men hier bedoelen met 'op zijn logische plaats' zetten?
 - Wat kunt u concluderen uit het feit dat reorganisatieruns veelvuldig moeten plaatsvinden?
 - Wat kunt u concluderen uit de in de laatste zinsnede vermelde noodzaak tot sorteren van het mutatiebestand?

Opgaven 7.2, 7.3 en 7.4 hebben betrekking op het voorbeeld van par.

7.1.4. Wij zullen daarbij gebruik maken van de volgende notaties:

t_0 : tijdstip vlak vóór het aanbrengen van de in 7.1.4 genoemde mutaties;

t_m : tijdstip vlak ná het aanbrengen van de mutaties.

- Op tijdstip t_0 moet het record met sleutelwaarde 518 worden ingevoegd. Hoeveel leesopdrachten en hoeveel schrijfoopdrachten zijn hiervoor nodig? Idem op tijdstip t_m .
 - Op tijdstip t_m moet het record met sleutelwaarde 520 worden gewijzigd. Hoeveel leesopdrachten en hoeveel schrijfoopdrachten zijn hiervoor nodig?
- 7.3. Is het voor deze I-S organisatie bezwaarlijk als in de trajecttabel (in variant A en B) voor iedere overloopketting het adres

van het record met de hoogste in plaats van de laagste sleutelwaarde in de ketting wordt genoteerd?

- 7.4. Bij het opzetten van het bestand wordt besloten in de hoofdtrajecttabel en in de trajecttabellen niet de hoogste maar de laagste sleutelwaarde van het desbetreffende traject op te nemen. Voer nu vanaf tijdstip t_0 de in 7.1.4 genoemde mutaties uit en geef met name weer de inhoud van het desbetreffende traject en van de trajecttabel na het uitvoeren van de mutaties.
- 7.5. Geef de programmaschetsen analoog aan die van par. 7.1.5 maar daarbij uitgaande van het gegeven in opgave 7.4.
- 7.6. Ga na, voor elk van de varianten A, B en C in paragraaf 7.1.4, dat gebruik en onderhoud van het bestand ook mogelijk is wanneer in een of meer primaire trajecten, wegens voorafgaande verwijderingen, geen (geldige) records staan.

In opgaven 7.7, 7.8 en 7.9 worden programmaschetsen gevraagd en dient men uit te gaan van dezelfde specificaties als die in paragraaf 7.1.5.

- 7.7. Het sequentieel lezen van het bestand.
- 7.8. Het invoegen van één record.
- 7.9. Het sequentieel verwerken van een mutatiebestand dat alleen verwijderingen bevat.
- 7.10. Gegeven is een index-sequentieel georganiseerd bestand. Behalve het sleutelveld bevat elk record nog een ander veld, waarvan de waarde uniek is. Met andere woorden dit andere veld zou ook dienst kunnen doen als sleutelveld.
 - a. Welke voorziening kan worden getroffen, zodat het bestand sequentieel toegankelijk is op dit 'tweede sleutelveld'?
 - b. Geef een programmaschets voor het invoegen van een record tijdens een sequentiële verwerking
 1. op het sleutelveld;
 2. op het 'tweede sleutelveld'.
- 7.11.
 - a. Geef de inhoud van de eerste tabel, dus op relatief adres 0, van het voorbeeld in paragraaf 7.2.
 - b. Voor het bestand van tabel 1 in hoofdstuk 6 wordt een index-directe organisatie opgezet met de volgende specificaties:
 - het tabelbestand bestaat uit 10 tabellen
 - elke tabel kan maximaal 12 paren (sleutelwaarde, adres)

- bevatten, plus een paar met sleutelwaarde ∞ .
 - in het gegevensbestand liggen de records opgeslagen in de volgorde van tabel 1 van hoofdstuk 6
 - voor de conversie wordt gewerkt met sleutelwaarde mod 10.
- b. 1. Is er overloop te verwachten?
- b. 2. Komt er werkelijk overloop voor?
- b. 3. Geef de inhoud van de 8e tabel, dus met adres 7.
- 7.12. Geef een programmaschets voor elk van de volgende verwerkingen op een index-direct georganiseerd bestand:
- a. het zoeken en afdrukken van één record;
 - b. het verwijderen van één record.
- 7.13. a. Waarom hoeft men bij gebruik van B-bomen nooit het bestand te reorganiseren?
- b. Heeft het voordelen om de wortel van een B-boom ook permanent in het werkgeheugen te hebben, al kost dat weer een ruimte ter grootte van een knoop?
- c. Waarom legt men aan een B-boom de in 7.3.1 genoemde voorwaarden op?
- 7.14. Bouw zelf een aantal B- en B⁺-bomen voor $k=3$ en $k=7$ op met de eerste 50 sleutelwaarden van tabel 1 in hoofdstuk 6.

8 GEINVERTEERDE BESTANDEN

8.1 GEINVERTEERD BESTAND ALS ANDERS GESORTEERD BESTAND

Tot nu toe hebben we alleen gesproken over toegang tot een record via zijn sleutelwaarde. In paragraaf 2.3 werd er reeds op gewezen dat benadering soms ook moet kunnen plaatsvinden via een proces van opsporing (retrieval). In dit opsporingsproces tracht men records te vinden op grond van andere waarden dan sleutelwaarden. Geen van de besproken bestandsorganisaties is eigenlijk ingesteld op zo'n opsporingsproces. Er zal dan ook veelal niet anders op zitten dan alle records van een bestand te lezen, de desbetreffende velden te onderzoeken en de records die voldoen aan de retrieval-vraag weg te schrijven naar een andere plaats (of af te drukken). Op deze wijze moet het volledige bestand worden doorgewerkt, hetgeen voor zeer grote bestanden een tijdrovende en kostbare zaak kan worden. Dit geldt met name als een dergelijke retrieval-vraag niet eenmalig is maar herhaaldelijk voorkomt.

Dikwijls kan een zogeheten geïnverteerd bestand hier goede uitkomst bieden. Wat een geïnverteerd bestand is zal nu eerst aan de hand van een alledaags voorbeeld worden duidelijk gemaakt.

Als men, gegeven een naam (en adres), het bijbehorende telefoonnummer zoekt in een telefoonboek, is dit, zoals bekend, een eenvoudige zaak. Immers (per woonplaats) zijn de abonnees sequentieel geordend op achternaam en daarbinnen op straatnaam. Met een 'soort binaire zoek' is het telefoonnummer dan ook snel gevonden. Het wordt echter een moeilijke (voor plaatsen met veel abonnees zelfs een ondoenlijke) zaak om in hetzelfde telefoonboek naam en adres van een abonnee te vinden, als vooraf alleen het telefoonnummer bekend is. In feite zal zo'n opdracht tot gevolg hebben dat gemiddeld het halve abonneebestand moet worden doorzocht (hopend dat het gegeven nummer werkelijk bestaat). Als het gegeven nummer niet bestaat zal zelfs het gehele bestand worden doorzocht. Komt zo'n vraag herhaaldelijk voor, dan ligt het voor de hand het bestand van abonnees (ook) in de volgorde van telefoonnummers beschikbaar te hebben. Een (hulp)bestand in de volgorde van een niet-sleutel veld noemen wij een

geïnvverteerd bestand (inverted file).

Bij elk bestand zijn vele geïnvverteerde bestanden te bedenken. Wij zullen dit toelichten aan het voorbeeld van een personeelsbestand, waarvan wij in dit hoofdstuk verder op meer plaatsen gebruik zullen maken.

Wij gaan uit van een personeelsbestand, waarbij in ieder record de volgende velden voorkomen:

nrnr	: registratienummer (sleutel)
nm	: naam
adr	: adres
wpl	: woonplaats
anr	: afdelingsnummer
vgc	: vakgroepcode
gbd	: geboortedatum (jaar, maand, dag)
bs	: burgerlijke staat (ongehuwd, gehuwd, weduwnaar/weduwe, gescheiden)
ak	: aantal kinderen

Dit bestand kan uiteraard sequentieel geordend zijn op (de sleutel) nrnr. Men kan dan een geïnvverteerd bestand I1 maken door dit bestand te sorteren op woonplaats en daarbinnen op achtereenvolgens adres en naam (aannemende dat de combinatie naam, adres, woonplaats ook sleutel kan zijn). Een ander geïnvverteerd bestand I2 zou kunnen zijn het bestand gesorteerd op vakgroepcode en daarbinnen op registratienummer.

8.2 GEINVERTEERD BESTAND ALS VERZAMELING VAN TABELLEN

Met het geïnvverteerde bestand I1 is het nu eenvoudig per woonplaats voorkomende adressen en namen op te sporen, terwijl met I2 op eenvoudige wijze de samenstelling van een vakgroep is te achterhalen.

Nu is men in veel gevallen niet geïnteresseerd in de samenstelling van alle vakgroepen of de adressen en namen in alle woonplaatsen, maar de samenstelling van een bepaalde vakgroep of de adressen en namen in één bepaalde woonplaats. In zo'n geval heeft men dan niet het gehele geïnvverteerde bestand nodig, maar alleen dat stuk dat betrekking heeft op die bepaalde vakgroep of woonplaats. Verder zal men niet voor al die gewenste volgorden van (delen van) het bestand ook evenzovele malen de records van het desbetreffende (deel van het) bestand in hun geheel kopiëren. Dit zou nl. een erg dure voorziening worden. Men kiest dan ook in zo'n geval voor een tabel met voldoende informatie per record om dit in het (slechts eenmalig opgeslagen) bestand snel te kunnen vinden. Daartoe kan het voldoende zijn alleen de sleutelwaarde op te nemen; ook opname van alleen het adres kan soms voldoende zijn. In de meeste

gevallen zal echter opname van sleutelwaarde en adres aan te bevelen zo niet noodzakelijk zijn. Deze noodzaak kan o. a. te maken hebben met voldoende controlemogelijkheden op fouten. Wij zullen nu verder veronderstellen dat in genoemde tabellen steeds sleutelwaarde en adres van elk record zijn opgenomen en dat elke tabel wordt afgesloten met de sleutelwaarde ∞ . Het bestand zal dus op een adresseerbaar medium moeten worden opgeslagen. Per veldwaarde, volgens welke men het bestand herhaaldelijk wil benaderen, kan nu zo'n tabel worden opgesteld. Let wel, dat er sprake moet zijn van herhaaldelijk benaderen, anders is de aanmaak van genoemde tabellen zinloos.

Als wij zo'n tabel een geïnvverteerd bestand noemen, dan gaat dit eigenlijk in twee opzichten niet op, want

- de tabel heeft slechts betrekking op een deel van het bestand,
- de tabel bevat slechts een (klein) deel van elk record.

Wij zullen overigens voor zo'n tabel de naam geïnvverteerd bestand niet gebruiken. Dit echter meer om de volgende reden. Wij willen tot uitdrukking brengen dat een verzameling tabellen betrekking heeft op een en hetzelfde bestand. Aan deze verzameling zullen wij dan (weliswaar enigszins op inconsequente wijze) de naam geïnvverteerd bestand geven en een tabel dan aangeven met: record van een geïnvverteerd bestand of (korter en nog ietsje inconsequenter) geïnvverteerd record. Wij zijn met opzet zo uitgebreid op deze naamgeving ingegaan. Immers de ervaring heeft geleerd dat daarmee onnodige misverstanden kunnen worden voorkomen.

Keren wij terug naar het voorbeeld van het personeelsbestand. Wij veronderstellen dat regelmatig te beantwoorden vragen worden gesteld die betrekking hebben op de volgende velden en bijbehorende waarden:

anr	:	5, 13, 22
vgc	:	C, H, S, T
gbd	:	> 541231
bs	:	og (ongehuwd)
combinatie	{	bs : gh (gehuwd)
van		ak : 0 (geen kinderen)

Vragen als bedoeld zijn bijvoorbeeld:

- vraag 1 : druk af registratienummer, naam, adres en woonplaats van personeelsleden die werken in afdeling 13.
- vraag 2 : druk af registratienummer en naam van de personeelsleden, die werken in afdeling 22 en behoren tot vakgroep H of T.
- vraag 3 : druk af naam, adres en woonplaats van de personeelsleden, die ongehuwd zijn of (gehuwd en geen kinderen en geboren na 1954).

In verband met de relevante waarden zal nodig zijn een geïnvverteerd bestand met tien records. Let wel, voor de combinatie (gehuwd, geen

kinderen) is volgens bovenstaande specificatie slechts één record nodig. Buiten deze tien records zal nog een aparte tabel nodig zijn om te kunnen vaststellen waar, gegeven veldnaam en veldwaarde, het bijbehorende geïnverteerde record is opgeslagen. In dit voorbeeld zou deze aparte tabel er als volgt uit kunnen zien:

veldnaam	veldwaarde	adres in geïnverteerde bestand
anr	5	1
anr	13	2
anr	22	3
vgc	c	4
vgc	h	5
vgc	s	6
vgc	t	7
gbd	541231	8
bs	og	9
bsak	gh0	10

Opmerking: een andere volgorde van de adressen in de laatste kolom is uiteraard ook mogelijk.

Wij hebben nu te maken met drie verschillende recordtypen, dus ook met (minstens) drie verschillende bestanden, en wel voor

- de records in het stambestand
- de records in het geïnverteerde bestand
- de aparte tabel, om te kunnen zoeken in het geïnverteerde bestand.

8.3 UITGEWERKT VOORBEELD

Wij zullen nu een programmaschets ontwikkelen voor de beantwoording van vraag 2.

Het is duidelijk dat wij drie geïnverteerde records nodig hebben, nl. die welke betrekking hebben op $\text{anr} = 22$ ($\text{ir}22$), $\text{vgc} = \text{H}$ (irh) en $\text{vgc} = \text{T}$ (irt).

Aangezien $\text{vgc} = \text{H}$ òf $\text{vgc} = \text{T}$ voldoende is, zullen wij het record $\text{ir}22$ als uitgangspunt nemen. Per doorgang van de herhalingsopdracht zullen wij dan de 'aan de beurt zijnde' sleutelwaarde in $\text{ir}22$ afhandelen. Bij deze afhandeling zal het dan wel nodig zijn de records irh en irt zover te inspecteren, dat een definitieve afhandeling inderdaad mogelijk is.

Bij de programmaschets zullen wij gebruik maken van de procedure $\text{zoekad}(\text{tr}, \text{veldnaam}, \text{waarde})$, die gegeven de veldnaam en de waarde met behulp van de (bovengenoemde) aparte tabel tr aan zoekad het

adres toekent van het geïnvverteerde record dat betrekking heeft op de gegeven veldnaam en waarde.

De programmaschets is nu als volgt:

```

begin type record strec(rnr,nm,data),
    paar(sl,verw);
    invrec[1:500] of paar;
    regel(naam,waarde,adres);
    tabrec[1:50] of regel;
    file
        stbest of strec,
        invbest of invrec,
        tabbest of tabrec
    endtype;
    var stb: stbest; invb: invbest; tb: tabbest
    endvar;
    ready(stb,invb,tb);

begin var sr: strec; ir22,irh,irt: invrec; tr: tabrec;
    adres,j22,jh,jt: integer; s22,sh,st: sl;
    gelijk: boolean           %gelijk is true als sleutelwaarde in
                                ir22 ook voorkomt in irh of irt%
    endvar;
    read(tb,1,tr);
    adres := zoekad(tr,anr,22); read(invb,adres,ir22);
    adres := zoekad(tr,vgc,h); read(invb,adres,irh);
    adres := zoekad(tr,vgc,t); read(invb,adres,irt);
    j22 := jh := jt := 1; s22 := ir22[1].sl; sh := irh[1].sl;
    st := irt[1].sl;
    while s22 <  $\infty$  and (sh <  $\infty$  or st <  $\infty$ )
        do gelijk := false;
        while sh < s22
            do jh := jh+1; sh := irh[jh].sl
            od;
            if sh = s22
                then gelijk := true
                else while st < s22
                    do jt := jt+1; st := irt[jt].sl
                    od;
                    gelijk := st = s22
                fi;
                if gelijk
                    then read(stb,ir22[j22].verw,sr);
                     print(sr.rnr,sr.nm)
                fi;
                j22 := j22+1; s22 := ir22[j22,1]
            od
        od
    end
end

```

Opmerking.

In het gedeelte A van voorgaand programma wordt onderzocht of de sleutelwaarde s22 ook voorkomt in irh of in irt. In dit programma wordt daartoe eerst irh onderzocht en eventueel daarna ook nog irt. In omgekeerde volgorde, eerst irt en dan eventueel irh, kan het uiteraard ook en het maakt voor de oplossing van de gestelde vraag geen verschil.

8.4 GEBRUIK VAN GEINVERTEERDE BESTANDEN

Het aantal records in het geïnverteerde bestand moet wel beperkt blijven, anders moet een onevenredig grote prijs ervoor betaald worden bij het onderhoud. Immers elke mutatie op het stambestand kan een verandering in het geïnverteerde bestand tot gevolg hebben. Het aantal records in het geïnverteerde bestand wordt dus eerder kritisch naarmate het bestand een hogere vervangingsgraad en/of verloop en/of groei kent. Voor het opstellen van programmaschetsen voor onderhoud zij verwezen naar de opgaven.

Geïnverteerde bestanden zijn een belangrijk (en onmisbaar) onderdeel voor huidige documentatiesystemen, waarin men op verschillende trefwoorden ingang wil hebben. Belangrijke toepassingen zijn te vinden in bibliografische en medische documentatie e.d.

8.5 THESAURI

Wij zullen hier nog wat nader ingaan op de retrieval-problemen bij literatuurverwijzingen, mede omdat op dit gebied nogal wat aparte terminologie wordt gebruikt. Het begint al met de keuze van sleutels of zoals ze in dit terrein meestal genoemd worden: *descriptoren*. Descriptoren zijn namelijk vaak synoniemen in die zin dat zij voor een retrieval een identieke functie hebben. Zo kunnen bijvoorbeeld bepaalde wiskundige artikelen geclassificeerd worden met de descriptoren: discrete wiskunde of combinatorische wiskunde of misschien ook coderingstheorie. Wanneer men nu bij een retrieval-vraag niet alle synoniemen voor een bepaalde descriptor opgeeft, is het duidelijk dat bij retrieval slechts een deel van de gewenste literatuur gevonden wordt.

Bij automatische retrieval-systemen wordt het synoniemenprobleem als regel opgelost met een zg. *thesaurus*, die hierop berust dat alle synoniemen verwijzen naar een standaardterm die op zijn beurt verwijst naar alle records waarop de standaardterm betrekking heeft. De gebruiker van een thesaurussysteem hoeft deze standaardterm dus niet te kennen of op te zoeken (vergelijk het bij bibliotheekkaartjes bekende systeem van 'zie ook...') om toch de goede antwoorden op zijn retrieval-vraag te krijgen (uiteraard verder aannemend dat de descriptoren goed gebruikt zijn en geen spelfouten worden gemaakt). Voorbeeld: de synoniemen A'dam, Amstelredam, Mokum, enz. verwijzen allemaal eerst naar Amsterdam.

Een ander probleem dat zich bij automatische retrieval-problemen kan voordoen is, dat een bepaalde descriptor met verschillende interpretaties gebruikt wordt. Soms is dit een blunder te noemen bij de keus van toegelaten descriptoren, maar in andere gevallen is het misschien niet te vermijden. Voorbeeld: woorden als systeem, data

set. Bij gebruik van zo'n descriptor zonder meer krijgt men uiteraard te veel niet-relevante literatuurverwijzingen. Wordt zo'n descriptor in een and-relatie met een andere descriptor gebruikt, dan wordt deze fout veelal vanzelf geëlimineerd (bijvoorbeeld data set en frequency geeft vrijwel zeker alleen literatuur over modems). Bij geraffineerde retrieval-systemen wordt de vrager aangespoord eerst nog een synoniem voor een door hem gebruikte descriptor te geven of uit een lijst van synoniemen een keus te doen (bijvoorbeeld bij data set moet hij kiezen uit modem of file).

OPGAVEN

- 8.1. Gegeven is een sequentieel georganiseerd bestand (opgeslagen op magneetband) van klanten. Elk klantenrecord bevat o. a. het sleutelitem klantnummer en de items woonplaats, groepscode en bezorgingscode.
Men wil nu naast het stambestand een geïnvverteerd bestand bouwen met de records:
ehv : bevattende sleutelwaarde en recordadres van alle klanten in Eindhoven,
gr12 : bevattende sleutelwaarde en recordadres van alle klanten in klantengroep 12,
b3 : bevattende sleutelwaarde en recordadres van alle klanten met bezorgingscode 3.
Overigens moet het klantenbestand sequentieel toegankelijk blijven.
Geef aan welke gegevensorganisatie nodig en voldoende is en welke verwerking moet worden uitgevoerd om het gevraagde te verwezenlijken. De verwerking in grote lijnen beschrijven.
- 8.2. Geef een programmaschets voor
 - a. vraag 1 van paragraaf 8.2;
 - b. vraag 3 van paragraaf 8.2.
- 8.3. Van het personeelsbestand, dat in dit hoofdstuk werd besproken, is gegeven dat het direct georganiseerd is met directe relatie. Van een bepaald personeelslid moeten afdelingsnummer en vakgroepcode veranderd worden. Daartoe worden opgegeven:
rnr : registratienummer van dit personeelslid
nanr : nieuw afdelingsnummer
nvgc : nieuwe vakgroepcode.
 - a. Welke veranderingen kunnen ten gevolge van deze mutatie moeten plaatsvinden?
 - b. Geef een programmaschets van deze mutatie, inclusief de mogelijke aanpassingen in het geïnvverteerde bestand.

9 KEUZE VAN BESTANDSORGANISATIE

In de voorgaande hoofdstukken werden verschillende vormen van bestandsorganisatie apart behandeld. In dit hoofdstuk zullen enkele gedachten ontvouwd worden over het onderwerp 'keuze van een bestandsorganisatie'.

Bij de keuze van de organisatie voor een bepaald bestand moet steeds het beoogde gebruik van dit bestand voorop staan. Hierbij moet echter worden bedacht, dat zo'n gebruik veelvuldig en gevarieerd kan zijn.

Bestanden vormen immers geen onveranderlijke verzamelingen van gegevens (waarden). Integendeel, deze gegevens (waarden) zullen dikwijls in hoge mate veranderlijk zijn. Men kan zelfs stellen dat het vaak juist de (sterk) veranderlijke items zijn, waarnaar in eerste instantie wordt gevraagd. In verband met dit veranderlijke, dynamische aspect van bestanden, is onderhoud (maintenance) noodzakelijk.

De dynamiek van een bestand kan worden vastgelegd met behulp van kentallen als de in hoofdstuk 2 genoemde bestandsverandering, bestandsverloop en bestandsgroei.

Naarmate de organisatie van een bestand ingewikkelder is en tegelijk de dynamiek sterker, zal sneller een zodanige ondoelmatigheid van sommige bestandsvormen kunnen optreden, dat veelvuldige reorganisatie nodig is. Kosten voor reorganisatie, vooral van omvangrijke bestanden, moeten niet worden onderschat.

In voorgaande hoofdstukken is er verder al menigmaal op gewezen, dat bij onderhoud (gemakkelijk) fouten kunnen binnensluipen. En wij bedoelden dan met name fouten in mutatiebestanden, die dan in stambestanden kunnen doordringen. Rekening houden met fouten maakt bestandsonderhoud ingewikkeld en duur.

De wijze waarop men gebruik wil maken van en men onderhoud wil plegen op een bestand is dus bepalend voor de keuze van bestandsorganisatie.

Hierbij is een ander, reeds dikwijls genoemd, kengetal van groot belang, namelijk de bestandsactiviteit. In hoofdstuk 2 werd er reeds op gewezen dat dit kengetal niet afhankelijk is van het bestand

alleen, maar ook van de programma's waarin dit bestand nodig is. In de loop van de tijd kan voor eenzelfde bestand in eenzelfde programma, de bestandsactiviteit wisselend zijn. Men hoede zich in dit opzicht voor het misverstand als zou de gemiddelde bestandsactiviteit van enige, laat staan doorslaggevende, betekenis zijn. Deze betekenis kan dit gemiddelde niet hebben, omdat hoge bestandsactiviteit om essentieel andere organisatie kan vragen dan lage bestandsactiviteit.

Bij gebruik en onderhoud is verder van belang, of toegang tot het bestand wordt gezocht op basis van sleutelwaarden of op basis van niet-sleutelwaarden. Beide gevallen zullen wij nu achtereenvolgens in beschouwing nemen. Wij beginnen met

I. TOEGANG OP BASIS VAN SLEUTELWAARDEN

Hierbij kan men dan onderscheiden:

- Ia. Bestanden met uitsluitend hoge bestandsactiviteit.
- Ib. Bestanden met uitsluitend lage bestandsactiviteit.
- Ic. Bestanden met zowel hoge als lage bestandsactiviteit.

Alvorens deze drie gevallen verder te bespreken, een enkel woord over wat onder hoge en lage bestandsactiviteit kan worden verstaan.

Het is zonder meer duidelijk dat wanneer alle records van een bestand nodig zijn, er sprake is van de hoogst mogelijke bestandsactiviteit, en wanneer slechts één record nodig is, er sprake is van de laagst mogelijke bestandsactiviteit. De 'hamvraag' is echter: Waar ligt de grens tussen hoog en laag? Over een juiste keus van deze grens bestaat nog grote onduidelijkheid. Deze onduidelijkheid is terug te voeren op het feit dat er nog maar zeer gebrekkig ontwikkelde optimaliseringsmodellen bestaan voor keuze van een bestandsorganisatie. Een vaak gehanteerde vuistregel is: bestandsactiviteit is hoog respectievelijk laag als deze meer respectievelijk minder dan 10% bedraagt. Nogmaals: niet meer dan een vuistregel, die echter in voorkomende gevallen goede diensten kan bewijzen.

In de voorgaande hoofdstukken is uitvoerig ter sprake gekomen dat de globale oplossingen voor de gevallen Ia, Ib en Ic zijn: sequentiële bestandsorganisatie, directe bestandsorganisatie en index-sequentiële bestandsorganisatie. Wij zullen de analyses, die leidden tot deze oplossingen, hier niet herhalen. Men dient er verder oog voor te hebben, dat naarmate de bestandsorganisatie in wezen ingewikkelder wordt (en zo is het in de volgorde a, b en c), ook de (nog verder te kiezen) varianten in aantal toenemen. Daarbij moet dan verder worden bedacht dat varianten, die leiden tot eenvoudiger gebruik, meestal tot minder eenvoudig onderhoud leiden. In het algemeen kan men namelijk als stelregel aanhouden: eenvoud van gebruik gaat gepaard met ingewikkelder onderhoud en omgekeerd. Denk maar aan voorbeelden als: een geordende ket-

ting is moeilijker te onderhouden dan een ongeordende ketting, maar een geordende ketting is eenvoudiger in het gebruik dan een ongeordende; consecutieve opslag kan tijdbesparend werken bij gebruik, maar het onderhoud tijdrovend maken.

Bij keuze van varianten is het dan ook zaak de relatieve kosten van gebruik en onderhoud zo goed mogelijk tegen elkaar af te wegen. Bovengenoemde stelregel geeft daarover geen uitsluitel. Ook zal men daarbij de frequentie van gebruik en onderhoud moeten betrekken. Zo zal het geval van veelvuldig onderhoud en matig gebruik leiden tot een variant met een eenvoudig onderhoud en minder eenvoudig gebruik.

Wij zullen nu bespreken de

II. TOEGANG OP BASIS VAN NIET-SLEUTELWAARDEN

Ook hier onderscheiden wij dan weer drie gevallen:

- IIa. Bestanden met uitsluitend hoge bestandsactiviteit.
- IIb. Bestanden met uitsluitend lage bestandsactiviteit.
- IIc. Bestanden met zowel hoge als lage bestandsactiviteit.

Ad IIa.

Keuze voor sequentiële bestandsorganisatie.

Ad IIb.

Hierbij dient verder onderscheid gemaakt te worden tussen een matig respectievelijk periodiek vereiste informatie. Dit leidt namelijk tot de keuze van sequentiële bestandsorganisatie zonder respectievelijk met een geïnverteerd bestand.

Ad IIc.

Ook hier moet ten aanzien van de gevallen van lage bestandsactiviteit hetzelfde onderscheid gemaakt worden als sub b, met dezelfde consequenties voor de te maken keuzen.

Na deze aparte behandelingen van toegang op sleutelwaarden en niet-sleutelwaarden, volgt nu hieronder tot slot een schema voor de combinatie van beiden.

Bestandsactiviteit bij toegang op niet-sleutel waarden Bestands- activiteit bij toegang op sleutelwaarden	Niet van toepassing	hoog	laag		gemengd	
			eenmalig gebruik	periodiek gebruik	eenmalig gebruik	periodiek gebruik
hoog	S	S	S	S + GB	S	S + GB
laag	D	IS	IS	D + GB	IS	IS + GB
gemengd	IS	IS	IS	IS + GB	IS	IS + GB

Opmerkingen.

1. D kan door ID worden vervangen wanneer de opslagcapaciteit een beperkende factor is.
2. Betekenis van de afkortingen:
S = sequentieel; D = direct; IS = index-sequentieel;
ID = index-direct; GB = geïnverteerd bestand.

OPGAVEN

- 9.1. Een gemeente wil gegevens over ingezetenen en woningen zodanig vastleggen in bestanden, dat op efficiënte wijze kan worden voldaan aan de volgende informatie-eisen:
 - I. Elke maand moet een lijst worden afgedrukt van alle leegstaande woningen, op alfabetische volgorde van adressen, met de volgende gegevens per woning: adres (= sleutel), bouwjaar, naam en adres eigenaar, inhoud, aantal vertrekkenden, soort woning (eengezins of flat), duurteklasse (code 1, 2, 3 of 4).
 - II. Elk half jaar moet een lijst worden afgedrukt van alle ingezetenen, die in het daaropvolgende half jaar de leeftijd van 18 jaar bereiken (in verband met persoonlijk woonrecht). De lijst moet geordend zijn naar administratienummer en per ingezetene moet op de lijst voorkomen: administratienummer (= sleutel), naam, adres, geboortedatum.
 - III. Op elk willekeurig moment beantwoording van de volgende vragen:
 - a. Geef het aantal leegstaande woningen per wijk. (De gemeente heeft 20 wijken.)
 - b. Geef van een woning met een bepaald adres de volgende gegevens: naam en adres eigenaar, bewoningstoestand (wel of niet bewoond), bewoningsdoel (voor alleenstaande of voor gezin zonder kinderen of voor gezin met 1 of 2

kinderen of voor gezin met meer dan 2 kinderen), onderhoudstoestand (code 1, 2 of 3).

- c. Geef de adressen waarop een bepaalde persoon gedurende een bepaalde periode (bijvoorbeeld 1970-1980) heeft gewoond en geef bij elk van deze adressen: begin- en einddatum van bewoning, duurteklasse en soort woning en aantal vertrekken van de woning (men mag hierbij veronderstellen dat de duurteklasse van een woning in de loop van de tijd niet verandert).
- d. Geef een lijst van alle leegstaande woningen (op adresvolgorde) van duurteklasse 1 voor elk van de wijken 7, 8, 15 en 19. Op elk van deze vier lijsten de volgende gegevens per woning: adres, duur leegstand, aantal vertrekken, bouwjaar.
- e. Geef een lijst van alle alleenstaande ingezetenen (op volgorde van administratienummer), die woningzoekend zijn. Per ingezetene vermelden: administratienummer, naam, adres, woningklasse (code a, b, c of d, aangevend waarvoor ingezetene in aanmerking komt). Een ingezetene kan slechts voor één woningklasse in aanmerking komen.

Vraag. Geef de recordstructuur van het woningen- en het ingezetenenbestand. Voor elk bestand daarbij aangeven welke velden per record moeten voorkomen. Ook voor elk bestand aangeven welke bestandsorganisatie en welk opslagmedium nodig is (inclusief eventuele additionele voorzieningen). Met een korte en duidelijke toelichting weergeven hoe u tot de door u voorgestelde structuur en organisatie komt op grond van bovengenoemde informatie-eisen. Andere dan genoemde eisen spelen geen rol.

- 9.2. Een uitzendbureau ten behoeve van de verzorgende sector, zendt personen uit voor bepaalde functies, in bepaalde instellingen, in bepaalde plaatsen. Het gaat om de volgende functies, instellingen en plaatsen:

functies	:	verpleegkundige A
		verpleegkundige B
		verpleegkundige C
		ziekenverzorg(st)er
		huishoudelijke dienst
instellingen	:	ziekenhuis
		verpleegtehuis
		bejaardentehuis
		dienstencentrum
plaatsen	:	Eindhoven Nuenen
		Geldrop Son
		Heeze Veldhoven
		Waalre

Een persoon kan zich bij het bureau voor precies één functie laten inschrijven, echter wel voor meer dan één instelling in meer dan één plaats. Het uitzendbureau wil nu de gegevens van de ingeschreven personen vastleggen in één of meer bestanden (en eventuele hulpbestanden). Per ingeschreven persoon moeten de volgende gegevens worden vastgelegd: registratienummer (= sleutel), NAW, toewijsbare functie, toewijsbare instellingen en plaatsen, wel of niet uitgezonden. Met dit bestand (deze bestanden) moet aan de volgende eisen worden voldaan:

- I. Op een willekeurig moment antwoord op de volgende vragen:
 - a. Is een bepaalde persoon, waarvan het registratienummer wordt opgegeven, wel of niet uitgezonden.
 - b. Geef het aantal toewijsbare, niet-uitgezonden personen voor een bepaalde functie in een bepaalde instelling in een bepaalde plaats.
 - c. Geef een lijst met registratienummer en NAW van alle toewijsbare, niet-uitgezonden personen voor een bepaalde functie in een bepaalde instelling in een bepaalde plaats.
- II. Op elk willekeurig moment invoegen van nieuw ingeschreven personen of veranderen van gegevens van ingeschreven personen of verwijderen van ingeschreven personen.
- III. Per week een lijst produceren met de gegevens van alle ingeschreven personen. Deze lijst moet per functie worden weergegeven en binnen elke functie met oplopend registratienummer.

Vraag. Welke bestandsorganisatie is het meest geschikt voor bovengenoemde persoonsgegevens. Bij uw antwoord kort en duidelijk aangeven hoe aan elk van bovengenoemde eisen op geschikte wijze wordt voldaan. Andere dan genoemde eisen spelen geen rol.

10 DATABASES

10.1 INLEIDING

In de voorafgaande hoofdstukken 2 t/m 9 is in hoofdzaak aandacht geschonken aan het mechanisme van de opslagstructuren, behorend bij verschillende vormen van bestandsorganisatie. Ook is daarbij besproken waarom een bepaalde vorm gekozen wordt bij een gegeven bestand, waaraan door de gebruiker bepaalde eisen ten aanzien van toegankelijkheid worden gesteld. Zoveel mogelijk is daarbij ook onderzocht wat men over de toegangstijd van een record kan zeggen.

Zaken die echter nauwelijks besproken zijn (afgezien misschien van enkele uiterst geschematiseerde praktijkprobleempjes in de opgaven) zijn o. a.:

- a. Hoe wordt vastgesteld dat voor een (meestal onvolledig beschreven) praktijkprobleem enerzijds bepaalde gegevens relevant zijn en dus moeten worden opgenomen in het 'probleem model' en anderzijds andere gegevens niet relevant zijn (voor bijv. rapportages). Deze vaststelling volgt uit de informatieanalyse, hetgeen, zoals wij reeds eerder vermeldden, buiten het bestek van dit boek valt.
- b. Is eenmaal voor elk probleem vastgesteld welke gegevens relevant zijn, dan kan vervolgens de vraag worden gesteld of voor een aantal 'verwante' problemen evenzovele verschillende (logische) representaties moeten worden opgesteld en onderhouden (met uiteraard vastlegging van dezelfde gegevens op meerdere plaatsen), dan wel één alles omvattende representatie (een database) meer op zijn plaats is. In dit laatste geval zal men voor een bepaalde toepassing slechts van een gedeelte van de totale database gebruik maken, maar voor alle toepassingen samen van de gehele database.
- c. Bij het opstellen van een database dient men te bedenken welke overwegingen een rol spelen bij het 'verdelen' van de gegevens over de verschillende onderdelen van de database. Elk onderdeel kan dan worden beschouwd als een bestand, echter zo dat ook verbanden tussen bestanden kunnen zijn vastgelegd. In feite komt deze verdeling dus neer op het bepalen van de (logische) record-structuur en van de (logische) verbanden tussen de records. Men

spreekt in dit verband dan ook wel van het bepalen van de structuur van de database.

- d. Hoe moet een databasestructuur, als genoemd sub c, worden beschreven? En op welke wijze kan een database worden onderhouden en toegankelijk gemaakt voor alle 'belanghebbende' gebruikers?
- e. Nauw samenhangend met het voorgaande is de vraag of verandering in de databasestructuur (op logisch en/of opslag en/of fysiek niveau) kan plaatsvinden zonder verandering in toepassingsprogramma's. In dit geval spreekt men (meestal overigens op onduidelijke wijze) over 'data independence' van de toepassingsprogramma's. Met de bestandsorganisatiemethoden is data independence slechts tot op zekere hoogte te realiseren.

De hier genoemde en nog veel andere vragen worden in de overvloedige database-literatuur van de laatste tien à vijftien jaar besproken. We zullen in deze inleidende paragraaf 'in eerste ronde' op de zojuist gestelde vragen ingaan, zonder daarbij uiteraard volledigheid te willen pretenderen. In de volgende paragrafen zullen slechts enkele aspecten uitgebreider worden behandeld.

Ad a. Zoals reeds opgemerkt, zullen we hier niet nader ingaan op de informatieanalyse, die moet opleveren welke gegevens relevant zijn. Wij willen wel benadrukken dat dit altijd het uitgangspunt moet zijn voor gegevensstructurering. Het is helaas geen uitzondering dat relevante gegevens ontbreken en/of niet-relevante gegevens onnodig beslag leggen op de faciliteiten.

Ad b. De volgende overwegingen hebben ertoe geleid dat men nu in vele gevallen (in principe) streeft naar een database-opzet voor een bij elkaar behorende groep van toepassingen, bijv. van één onderneming.

- Met aparte, per toepassing gecreëerde, bestanden stuit men o. a. op de volgende problemen:
 - . in feite slecht toegankelijke gegevens, nl. alleen bereikbaar via speciaal voor die bestanden geschreven programma's
 - . meervoudige opslag van dezelfde gegevens in de verschillende bestanden (redundantie); daardoor gevaar voor inconsistentie van gegevens, die in verschillende bestanden identiek zouden moeten zijn; daaruit voortvloeiende inconsistentie in rapportages.
- De problemen van security (beveiliging van gegevens tegen verlies of verminking) en privacy (beveiliging tegen gebruik van gegevens door daartoe niet bevoegden) zijn beter te overzien (daarmee overigens nog niet eenvoudig op te lossen) met één samenhangende database.

Bij bovenstaande overwegingen moet wel het volgende worden opgemerkt. De aantoonbare directe kosten van verwerking van database-

systemen m.b.v. een computer zijn als regel vaak nog zoveel hoger dan die van conventionele systemen (met gebruikmaking van 'gewone' programmeertalen) dat zij de (moeilijk in geld uit te drukken) voordelen nu nog overtreffen.

Ad c. Op de (logische) structuur van een database zullen we in de volgende paragrafen wat nader ingaan.

Ad d. In het kader van de bestandsorganisatie wordt de gegevensstructuur beschreven als onderdeel van het toepassingsprogramma en dus met behulp van de programmeertaal, die voor het toepassingsprogramma wordt gebruikt. (Zo bevatten in een COBOL-programma de DATA DIVISION en de ENVIRONMENT DIVISION de logische en opslag-, resp. (gedeeltelijk) de fysieke structuurbeschrijving.)

De beschrijving van de gegevensstructuur voor een database vindt plaats met behulp van een aparte 'Data Definition Language (DDL)'. Bij deze beschrijving, ook wel Schema genoemd, wordt niet gedacht aan een fysieke structuur, maar wel aan de logische en (ten dele) de opslagstructuur. In de paragrafen 10.2, 10.3 en 10.4 zullen verschillende DDL's worden besproken.

Voor een toepassing zal het als regel niet nodig zijn dat men de totale database, dus het totale Schema, ter beschikking krijgt. Daarom worden dikwijls per toepassing deelbeschrijvingen gedefinieerd; deze noemt men dan Subschema's. Aangezien een database bedoeld is voor meer gebruikers, dient het beheer van een database te worden toevertrouwd aan een centrale persoon of instantie, de zogeheten 'Data Base Administrator' (DBA). Deze DBA is dan verantwoordelijk voor het opstellen van het Schema en van de Subschema's en voor security en privacy aspecten.

Om gebruik te maken van gegevens uit een database moet men kunnen beschikken over een zogeheten 'Data Manipulation Language (DML)'. Met behulp van een DML kan een gebruiker gegevens ophalen uit de database en/of gegevens opslaan in de database. Zo'n DML kan deel uitmaken van een reeds bestaande 'gewone' programmeertaal, bijv. COBOL. Men noemt dan de bestaande taal een 'host language', die uitgebreid is met de nodige DM-taalelementen.

Is een DML geen onderdeel van een host language, dan spreekt men van een 'self contained' DML. In paragraaf 10.4 wordt kort ingegaan op data manipulation languages, en in het bijzonder op zogeheten query languages (vraagtaalen).

Voor de uitvoering van een DML-opdracht, o.a. voor het ophalen c.q. opslaan van gegevens is speciale programmatuur nodig, het z.g. Data Base Management System (DBMS). Zo'n DBMS is een programmapakket met een niet ongebruikelijke orde van grootte van 50 K woorden. Elke gebruiker communiceert dan via het DBMS met de database (zie fig. 10.1).

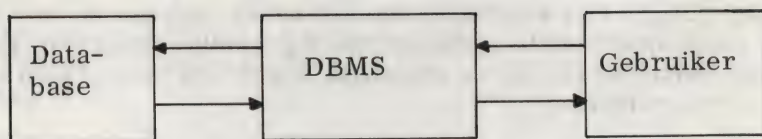


Fig. 10.1. Plaats van DBMS

Om de rol van de drie componenten van fig. 10.1 nog wat meer (ofschoon nog zeer schematisch) uit de verf te laten komen is in fig. 10.2 in grote lijnen weergegeven hoe een gegevensaanvraag van een programma via het DBMS tot stand komt. Hierbij is het volgende op te merken

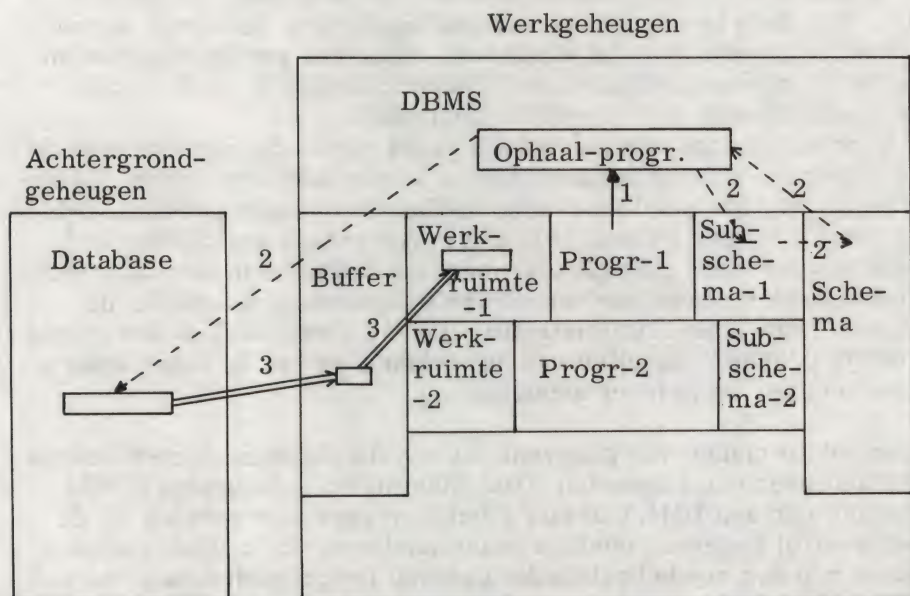


Fig. 10.2. Ophalen van gegevens uit een Database

- Pijl 1 : Ophaalopdracht uit programma 1 wordt met vermelding van de subschema-naam van de gegevens doorgegeven aan DBMS.
- Pijlen 2 : M.b.v. subschema en schema wordt de plaats van de gevraagde gegevens in de database bepaald.
- Pijlen 3 : De gegevens worden via de bufferruimte getransporteerd

naar de werkruimte van het aanvragende programma. Voor dit transport wordt meestal gebruik gemaakt van de 'access procedures' van het 'Operating System' van de installatie. In feite is het DBMS ook niets anders dan een speciale 'uitbreiding' van het operating system.

Er zij nogmaals op gewezen dat bovenstaande beschrijving van de (centrale) rol van een DBMS zeer summier en schematisch is.

Ad e. Over 'data independence' is al veel geschreven en dit niet altijd leidend tot een duidelijk begrip van wat het inhoudt.

Laten wij in de eerste plaats voorop stellen dat data independence uiteraard niet kan betekenen dat er helemaal geen verband bestaat tussen bijv. de structuurbeschrijving van de gegevens in een schema en de fysiek opgeslagen gegevens in een database. Dit verband moet bestaan omdat anders de gehele gegevensbeschrijving zinloos zou zijn. Een andere zaak is echter wie zich om dit verband dient te bekommeren, of bijv. een gebruiker moet weten hoe dit verband wordt geëffectueerd. Als wij dan (terecht) zeggen dat dit niet de taak dient te zijn van een gebruiker en dat de gebruiker met die zorg dan ook niet geconfronteerd dient te worden, geven wij daarmee de juiste draagwijdte van het begrip data independence aan. Op deze wijze gezien is de mate van data independence afhankelijk van de mate waarin de gebruiker geen aanwijzingen hoeft te geven over de (fysieke) opslag van gegevens en van de mate waarin verandering in (fysieke) opslag irrelevant is voor de gebruiker (programma's).

Naast deze onafhankelijkheid van de (fysieke) opslag (ook wel verticale onafhankelijkheid genoemd), is er ook nog de onafhankelijkheid van andere gebruikers (horizontale onafhankelijkheid), die tot uitdrukking komt in het opzetten van een eigen gebruikersschema, nl. het reeds eerder genoemde subschema naast het totale schema.

Met data independence wordt elke gebruiker (terecht) vrijgesteld van zorgen, die niet de zijne zijn (horizontale onafhankelijkheid) en van zorgen, die voor iedere gebruiker hetzelfde zijn en dus gemeenschappelijk moeten worden opgelost (vertikale onafhankelijkheid). Voor deze data independence moet echter wel een prijs worden betaald in de vorm van programmatuurvoorzieningen (dikwijls interfaces genoemd). In feite worden deze voorzieningen getroffen binnen het kader van het DBMS. Of liever, dienen te worden getroffen, want volledige data independence wordt (nog) door geen enkel DBMS geleverd (te hoge prijs!).

Tot slot van deze inleiding nog een enkel woord over bestaande, c.q. voorgestelde Data Base Systemen. In de loop der (laatste 15) jaren zijn diverse systemen ontwikkeld (zie Infotech 'State of the Art Report Data Base Systems', 1975), waarvan we hier noemen

- Adabas
- DMS 1100
- IDMS
- IMS
- Systeem 2000
- TOTAL

In de veelheid van systeemvoorstellen en daarmee samenhangende taalvoorstellen is getracht eenheid te brengen door middel van een standaardisatievoorstel. Het CODASYL Systems Committee liet daartoe een eerste voorstel het licht zien in 1971, gevolgd door herziene versies in 1973 en de jaren daarna. Door sommigen wordt wel min of meer ingespeeld op dit voorstel. Toch kan zeker niet gesproken worden van ruime erkenning. Enerzijds is dit te wijten aan de tegenstand van (software) fabrikanten die in standaardisatie een bedreiging zien van de eigen database-systemen. Anderzijds zouden minder goede aspecten van dit CODASYL standaardisatievoorstel zelf wel eens de grootste belemmering kunnen vormen voor erkenning op grote schaal.

Er is evenzo, sinds 1972, door ANSI (American National Standards Institute) via het zogeheten SPARC (Standards Planning and Requirements Committee) veel werk verricht om te komen tot standaardisatie van systemen voor databases. ANSI beoogt overigens zelf geen talen zoals een DDL te ontwikkelen.

10.2 DE LOGISCHE STRUKTUUR VAN EEN DATABASE. NORMALISATIE

In hoofdstuk 2 werd, aan het einde van paragraaf 2.1, reeds in het vooruitzicht gesteld dat in dit hoofdstuk over databases nader zou worden ingegaan op de logische structuur van gegevens. Met name zullen wij de zogeheten normalisatie bespreken waarmee systematisch kan worden vastgelegd welke attributen bij welke objecten kunnen behoren. Wij zullen hierbij uitgaan van het volgende objecttype (zie voor methode van beschrijving paragraaf 2.1):

- (0) leverancier(lnr, lnaam, lwoonpl, gcode,
 R. G. art-lev(anr, anaam, prijs),
 R. G. lev-proj(pnr, pnaam, R. G. proj-art(anr, anaam,
 hoev)))

waarbij:

- lnr : leveranciersnummer (uniek voor een leverancier)
 gcode : gemeentecode
 art-lev : de artikelen die een leverancier kan leveren
 anr : artikelnummer (uniek voor een artikel)
 lev-proj : de projecten, waaraan een leverancier artikelen levert

pnr : projectnummer (uniek voor een project)
proj-art : de benodigde artikelen per project; hieruit is als regel
niet af te leiden welke artikelen door een bepaalde leverancier aan een project worden geleverd
hoev : aantal dat een project van een bepaald artikel nodig heeft

De overige, niet-toegelichte, attributen namen spreken voor zichzelf.

Om redenen die verderop pas goed duidelijk kunnen worden is deze beschrijving aangegeven met het getal 0.

Om de precieze betekenis van de repeating group proj-art nader toe te lichten het volgende getallenvoorbeeld.

De leverancier met nummer L25 heeft de volgende artikelen in zijn assortiment (de artikelnummers worden gegeven) a3, a7, a12 en a37. Leverancier L25 levert o. a. aan project met nummer p17. Voor dit project p17 zijn nodig de artikelen met nummers a5, a12, a37 en a81. Uit dit gegeven volgt

wel dat L25 van a12 òf van a37 minstens een exemplaar aan p17 moet leveren,

niet dat L25 aan p17 alle benodigde exemplaren van a12 en a37 levert.

(Voor het geval dat dit laatste wel zou gelden zij verwezen naar opgave 9.)

Al met al is daarmee het objecttype leverancier van een wat uitzonderlijke structuur, maar dit is met opzet gedaan om daarmee de 'normalisatiestappen' duidelijk te kunnen laten zien.

Gegevens kunnen nu volgens de structuur van beschrijving 0 worden opgeslagen. We dienen echter wel te bedenken dat een structuur ontworpen wordt ten dienste van efficiënt gebruik en correct onderhoud. Gebruik van bovenstaand objecttype zal dan, zoals bij enig nadenken blijkt, leiden tot veel onnodig zoek en onderhoud en tot inconsistenties (d. w. z. gegevens die identiek moeten zijn, zijn het vaak niet). Deze moeilijkheden komen voort uit het feit, dat bovenstaande structuur zeer redundant is ten gevolge van de vele repeating groups.

Om redelijk gebruik en onderhoud van gegevens te waarborgen had men allang intuïtief begrepen (en ernaar gehandeld) dat al te gekke structuren zoals hierboven als regel moesten worden vermeden. Door Codd is de eerste stoot gegeven tot een meer formele benadering van structurering. Hij spreekt daarbij van normaliseren en onderscheidt voorts de eerste, tweede en derde normaalvorm.

Een objecttype is in de eerste normaalvorm, als er geen repeating groups in voorkomen. Om dit in ons voorbeeld te bereiken, zullen we beschrijving 0 moeten vervangen door bijvoorbeeld de volgende:

- (1) leverancier(lnr, lnaam, lwoonpl, gcode)
 art-lev(lnr, anr, anaam, prijs)
 proj-lev(lnr, pnr, pnaam)
 proj-art(pnr, anr, anaam, hoev)

Hierbij geven wij met onderstreping aan welk attribuut of welke combinatie van attributen als sleutel optreden.

In plaats van één objekttype hebben wij nu weliswaar vier objekttypen, maar wel met veel minder redundantie in de gegevensopslag. Overigens zit in deze structuur nog iets onbevredigends, nl. dat anaam resp. pnaam nog redundant voorkomen, hetgeen bij mutaties tot inconsistenties kan leiden (bedenk hoe). De reden hiervan is, dat deze attributen niet volledig afhankelijk zijn van de bijbehorende sleutels (lnr, anr), (pnr, anr) resp. (lnr, pnr) maar slechts van een gedeelte der sleutels en wel van anr resp. pnr.

Uitschakeling van deze 'niet-volledige' afhankelijkheden voert tot de tweede normaalvorm. In ons voorbeeld:

- (2) leverancier(lnr, lnaam, lwoonpl, gcode)
 artikel(anr, anaam)
 projekt(pnr, pnaam)
 art-lev(lnr, anr, prijs)
 proj-art(pnr, anr, hoev)
 proj-lev(lnr, pnr)

Met deze zes objekttypen (en nog minder redundantie) hebben wij nu bereikt dat alle niet-sleutel-attributen volledig afhankelijk zijn van de totale sleutel. Er zit echter nog een mogelijkheid van redundantie in, als nl. een niet-sleutel-attribuut afhankelijk is van een ander niet-sleutel-attribuut (of van een combinatie van niet-sleutel-attributen). Wij veronderstellen nu dat dit geldt voor het attribuut gcode en wel dat dit afhankelijk is van lwoonpl. De redundantie die door deze afhankelijkheid ontstaat, is bijzonder vervelend als eens gcode verandert, maar kan worden geëlimineerd door overgang op de derde normaalvorm. In ons voorbeeld:

- (3) leverancier(lnr, lnaam, lwoonpl)
 artikel(anr, anaam)
 projekt(pnr, pnaam)
 code(woonpl, gcode)
 art-lev(lnr, anr, prijs)
 proj-art(pnr, anr, hoev)
 proj-lev(lnr, pnr)

Met deze zeven objekttypen is de gegevensstructuur dus nu in de derde normaalvorm (3NF). Het startpunt in ons voorbeeld was een onge-normaliseerd objekttype. Daarom gaven wij deze beschrijving ook de 'index' 0 mee, om aan te geven dat deze in de 'nulde' normaalvorm stond, dus ongenormaliseerd was. Om een gegevensstructuur in de

derde normaalvorm te krijgen is het niet (per sé) nodig eerst de eerste en tweede normaalvorm expliciet op te schrijven. Normaliseren (tot en met de derde normaalvorm) is uiteraard alleen mogelijk, als (via voorafgaande informatie-analyse) duidelijk de betekenis en het onderlinge verband der relevante objecten en attributen vastligt. De derde normaalvorm is dus niet bepalend voor de betekenis der gegevens maar dient er een duidelijke weergave van te zijn.

In een gegevensstructuur zijn verschillende klassen van verbanden mogelijk tussen objecttypen. Wij onderscheiden met name de verbanden

$n : 1$: zoals tussen art-lev en leverancier, d. w. z. bij een
(veel : een) leverancier-occurrence horen diverse art-lev-occurrences en bij een art-lev-occurrence hoort precies één leverancier-occurrence.

$n : m$: zoals tussen leverancier en artikel, d. w. z. een leverancier kan diverse artikelen leveren en een artikel kan door verschillende leveranciers worden geleverd.

Bij een genormaliseerde (3NF) gegevensstructuur liggen deze verbanden vast door de objecttypen. Zo geeft het objecttype art-lev het $n:m$ -verband weer tussen leverancier en artikel.

Om het gegeven voorbeeld nog wat aanschouwelijker in beeld te brengen volgen hier nu tabellen met mogelijke occurrences van de zeven objecttypen.

leverancier

<u>lnr</u>	<u>lnaam</u>	<u>woonpl</u>
L3	Jansen	Eindhoven
L7	Pieters	Amsterdam
L25	Adams	Rotterdam
L53	Smits	Eindhoven
L81	Jongen	Helmond

artikel

<u>anr</u>	<u>anaam</u>
a1	regelaar
a3	trap
a5	klem
a7	wiel
a12	kruk
a24	as
a37	schaar
a81	balk
a90	hoekrek

projekt

<u>pnr</u>	<u>pnaam</u>
p10	roodkoeler
p17	witkoeler
p23	witkar
p47	roodkar

<u>code</u>			<u>art-lev</u>		
<u>woonpl</u>	<u>gcode</u>		<u>lnr</u>	<u>anr</u>	<u>prijs</u>
Amsterdam	10		L3	a5	10
Rotterdam	10		L3	a24	150
Utrecht	15		L7	a1	25
Eindhoven	20		L7	a12	15
Haarlem	20		L7	a37	98
<u>proj-art</u>			L7	a81	230
<u>pnr</u>	<u>anr</u>	<u>hoev</u>	L25	a3	225
			L25	a7	160
p10	a1	3	L25	a12	20
p10	a5	30	L25	a37	90
p10	a7	5	L53	a3	225
p10	a24	10	L53	a24	145
p17	a5	10	L81	a1	30
p17	a12	3	L81	a3	220
p17	a37	5	L81	a24	120
p17	a81	4	L81	a81	340
p23	a7	5	<u>proj-lev</u>		
p23	a12	1	<u>lnr</u>	<u>pnr</u>	
p23	a24	4	L3	p10	
p23	a90	6	L25	p10	
p47	a7	5	L81	p10	
p47	a12	2	L3	p17	
p47	a24	4	L25	p17	
p47	a81	6	L7	p17	
			L25	p23	
			L53	p23	
			L81	p23	
			L3	p47	
			L25	p47	
			L7	p47	
			L53	p47	
			L81	p47	

Wij willen deze paragraaf met de volgende opmerkingen besluiten.

- Het is gebleken dat bij werkelijke toepassingen het niet altijd eenvoudig is de logische gegevensstructuur in 3NF te brengen. Dit heeft waarschijnlijk te maken met het feit dat voor werkelijke toepassingen vele afhankelijkheden moeten worden 'ontward'.
- Uit studies van de laatste jaren is gebleken dat 3NF niet in alle gevallen voldoende is om tot een duidelijke gegevensstructuur te komen. In dit boek gaan wij hier verder niet op in.

10.3 DATABASE MODELLEN

In de loop van de laatste vijftien jaren zijn er diverse zienswijzen ontstaan, volgens welke men de database problematiek wil benaderen. Deze zienswijzen geeft men gewoonlijk aan met de naam *database model*. (Men voegt veelal het woord *model* toe om te benadrukken dat het om een door een waarnemer vastgelegde abstrahering van de realiteit gaat.) Gedachtig het spreekwoord 'zoveel hoofden, zoveel zinnen' zijn er eigenlijk heel veel database modellen. Er zijn echter drie modellen, die 'eruit springen'. Deze zijn

- het hiërarchische model
- het netwerk model
- het relationale model

Wij zullen nu een korte schets van elk van deze modellen geven. Daarbij zullen wij de 'historische' volgorde aanhouden. Immers met het hiërarchische model (IMS) werd in de zestiger jaren een der eerste pogingen ondernomen om verbanden tussen aparte bestanden in de gegevensbeschrijving op te nemen. Vervolgens werd vanaf het einde der zestiger jaren een poging ondernomen om met een groep van geïnteresseerden (Codasyl Task Group) tot algemene (standaardisatie) afspraken te komen ten aanzien van beschrijving en gebruik van bestanden met hun onderlinge verbanden. Deze groep werkte met het netwerk model. Tenslotte werd vanaf het begin der zeventiger jaren een poging ondernomen (onder aanvoering van de reeds eerder genoemde E. F. Codd) om de opzet van gegevensstructurering wetenschappelijk te onderbouwen en zo ontstond het relationele model.

De schets in dit hoofdstuk is slechts zeer summier en zeker onvoldoende om daarmee een evaluatie van de genoemde modellen te plegen. Voor een uitgebreide behandeling zij verwezen naar het boek van Date (zie literatuuropgave).

HIERARCHISCH MODEL

In het hiërarchisch model gaat men ervan uit, dat alleen hiërarchische verbanden tussen objecttypen kunnen voorkomen. In termen van (het einde van) de vorige paragraaf betekent dit alleen verbanden waarbij elk 'lager' gelegen objecttype ('child') in verband staat met een 'direct hoger' gelegen objecttype ('parent'). Over normalisatie laat dit model zich helemaal niet uit, evenmin als over regels voor het uitbeelden van $n : 1$ en $n : m$ verbanden.

Wij zullen dit model nu nader toelichten aan een voorbeeld, 'afgestemd' op het voorbeeld van de vorige paragraaf. Daarbij zullen wij gebruik maken van de gebruikelijke diagrammen in het hiërarchische model. Wij beginnen dan met een diagram op objecttype-niveau.

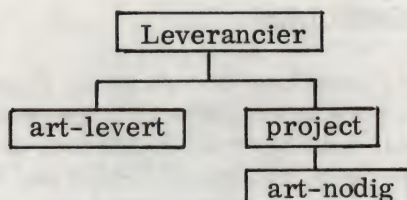


Fig. 10.3. Hiërarchische structuur (objekttype-niveau)

Met fig. 10.3 wordt bedoeld, dat bij elke leverancier nul, een of meer leverbare artikelen en nul, een of meer projecten in de database thuishoren, en verder dat bij elk project nul, een of meer te gebruiken artikelen in de database thuishoren. Wat echter precies per leverancier, artikel en project moet worden opgenomen in de database is uit het diagram van fig. 10.3 niet op te maken. Daartoe heeft men een diagram op attribuuttype-niveau nodig, zie fig. 10.4.

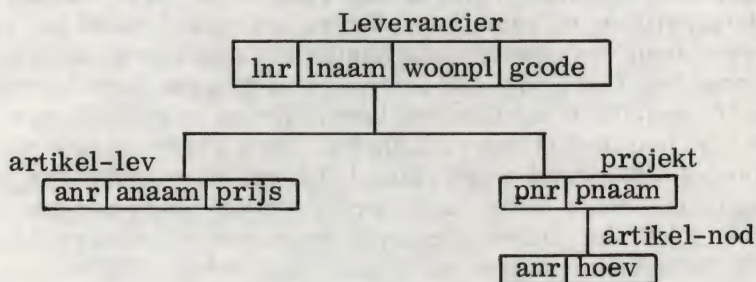


Fig. 10.4. Hiërarchisch model (attribuuttype-niveau)

Zoals gezegd speelt normalisatie geen enkele rol in dit model. Dus in artikel-nod had ook nog het attribuut anaam kunnen staan.

Voor een diagram op occurrence-niveau (daarbij uitgaande van de occurrences uit de vorige paragraaf), zie fig. 10.5.

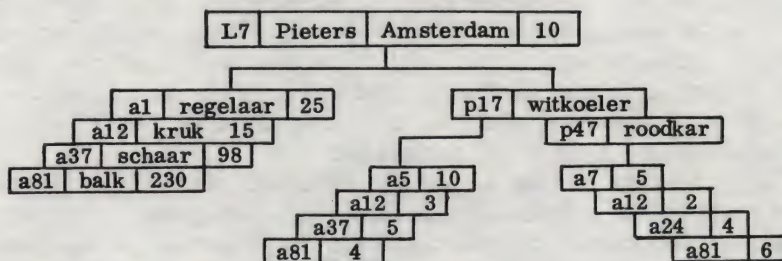


Fig. 10.5. Hiërarchische structuur (op occurrence-niveau)

Het aantal mogelijke occurrence diagrammen is in dit voorbeeld uiteraard gelijk aan het aantal leverancier-occurrences.

Diagrammen zijn een hulpmiddel om een gegevensstructuur (voor mensen) voor te stellen, maar het zijn geen beschrijvingsmiddelen om een gegevensstructuur voorstelbaar te maken voor de computer. De gegevensbeschrijving van fig. 10.4 zou in een taal à la IMS als volgt luiden:

```

1  DBD      NAME = LEV
2  SEGM     NAME = LEVERANCIER, BYTES = 75
3  FIELD    NAME = (LNR, SEQ), BYTES = 3, START = 1
4  FIELD    NAME = LNAAM, BYTES = 40, START = 4
5  FIELD    NAME = LWOONPL, BYTES = 30, START = 44
6  FIELD    NAME = GCODE, BYTES = 2, START = 73
7  SEGM     NAME = ART-LEVERB, PARENT = LEVERANCIER,
           BYTES = 37
8  FIELD    NAME = (ANR, SEQ), BYTES = 3, START = 1
9  FIELD    NAME = ANAAM, BYTES = 30, START = 4
10 FIELD    NAME = PRIJS, BYTES = 4, START = 34
11 SEGM     NAME = PROJECT, PARENT = LEVERANCIER,
           BYTES = 33
12 FIELD    NAME = (PNR, SEQ), BYTES = 3, START = 1
13 FIELD    NAME = PNAAM, BYTES = 30, START = 4
14 SEGM     NAME = ART-NODIG, PARENT = PROJECT,
           BYTES = 9
15 FIELD    NAME = (ANR, SEQ), BYTES = 3, START = 1
16 FIELD    NAME = HOEV, BYTES = 6, START = 4

```

Toelichting:

Bij regel 1: DBD: Data Base Description.

Bij regels 2, 7, 11, 14: Segment, min of meer equivalent met record.

In bovenstaande beschrijving zijn voorts nog allerlei specificaties betreffende de geheugenopslag weggelaten. De beschrijving van de gegevensstructuur in het hiërarchische model is dan ook, zoals bij zovele systemen, een mengelmoes van specificaties op hoog logisch en laag opslagniveau. Dit komt de doorzichtigheid niet ten goede. Verder houdt het model streng aan de in de beschrijving opgegeven structuur vast. Zelfs verwisseling van ART-LEVERB en PROJECT in fig. 10.3 zou al leiden tot een verplaatsing van de regels 7-10 na regel 16 in de beschrijving.

In feite ligt met de hiërarchische structuur ook de bereikbaarheid van de verschillende onderdelen van deze structuur vast. Zo is in het algemeen een occurrence van ART-NODIG alleen maar bereikbaar via zijn bovenliggende parents van PROJECT en LEVERANCIER. In sommige implementaties van hiërarchische structuren is bereikbaarheid ook mogelijk los van de bovenliggende parents. In zulke

gevallen is dan echter sprake van een 'uitbreiding' van een hiërarchische structuur met eigenlijk wezensvreemde elementen.

Tot slot van deze paragraaf nog twee opmerkingen:

- Bij bestudering van bovenstaand voorbeeld zal het opvallen dat de geschetste structuur in feite niets anders is dan de ongenormaliseerde vorm, dus met repeating groups.
- Deze repeating groups kunnen in principe ook binnen het hiërarchische model worden buitengesloten. Dit zou echter leiden tot een verwrongen en onhandelbare structuur.

NETWERKMODEL

In het netwerkmodel is het begrip set zeer belangrijk. De benaming voor dit begrip is ongelukkig, zeker voor wiskundig geschoolden. Immers in de wiskunde heeft set de betekenis van verzameling, terwijl het netwerkmodel (althans volgens Codasyl-aanpak) hieraan de betekenis toekent van afbeelding (functie van N op 1). Een en ander zal uit het volgende duidelijk worden.

Een objekttype wordt in het netwerkmodel met een recordtype weergegeven. De verbanden tussen objekttypen worden met behulp van settypen weergegeven.

Volgens het Codasyl-voorstel is sprake van een set-type als een record-type als owner optreedt en een of meer andere recordtypes als member optreden. Per set horen bij een record van het owner-type nul, een of meer records van elk member-type. Omgekeerd hoort per set-type bij een record van een member-type echter precies één record van het owner-type. Uit de voorgaande regels volgt, dat tussen owner en member-records een 1 op N relatie ($N \geq 0$) toegestaan is, maar geen M op N relatie ($M \geq 2$). Zo kan het verband tussen leverancier en artikel (zie paragraaf 10.2) niet met één set-type worden weergegeven.

Een voorbeeld van een verband, dat met één set-type kan worden gerepresenteerd, is het volgende.

Een bedrijf heeft verschillende afdelingen. Elke werknemer hoort precies bij één afdeling. Dit verband kan nu worden gerepresenteerd met het set-type AFD-WERKN, waarbij het record-type AFDELING het owner-type is en het record-type WERKNEMER member-type. Een set-type kan met het diagram van fig. 10.6 worden weergegeven. Een diagram als in fig. 10.6 met record- en set-typen wordt ook wel een Bachman-diagram genoemd (naar C.W. Bachman, een van de grondleggers van het netwerkmodel). De pijl wijst van owner naar member-type. Naast de pijl staat de naam van het set-type.

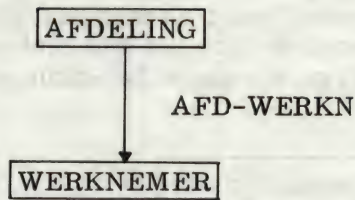


Fig. 10.6. Voorbeeld van een set-type

Een netwerk is de meest algemene vorm van een gegevensmodel. In feite kan in een netwerk elk record-type verbonden zijn met elk ander record-type. Of dit verband echter rechtstreeks via een set-type kan worden vastgelegd hangt van het soort verband af. Zo is de structuur in fig. 10.7 in het netwerkmodel niet toegestaan.

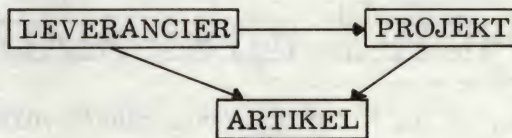


Fig. 10.7. Netwerk-structuur

Men stuit nl. op de moeilijkheid dat elk van de pijlen niet een 1 op N maar een M op N verband moet representeren. We zullen dan ook per verband een extra record-type moeten invoeren (zie fig. 10.8).

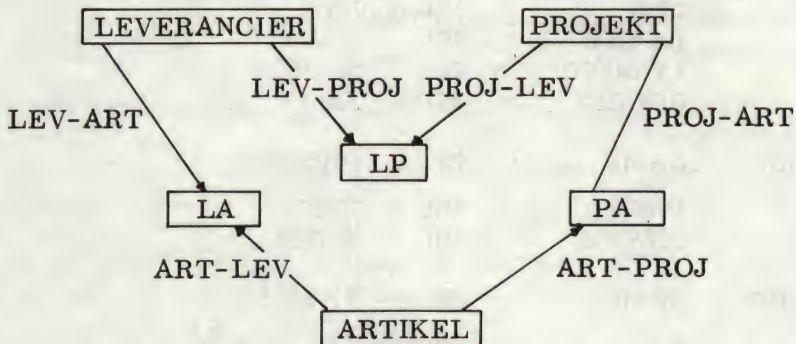


Fig. 10.8. Representatie van netwerk met set-types

In fig. 10.8 staan de namen van de zes set-typen bij de respectieve verbindingspijlen tussen owner en member-type. Op attribuut-type niveau zou het diagram er voor het stuk met de record-typen leverancier, projekt, LP en de set-typen Lev-proj en proj-lev als volgt uit kunnen zien:

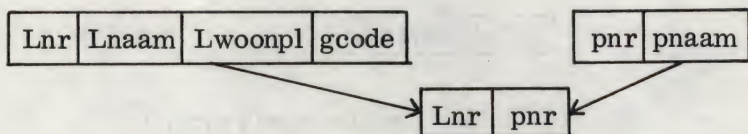


Fig. 10.9. Netwerk op attribuut-type-niveau

Tenslotte een diagram op occurrence-niveau (weer op basis van occurrences in paragraaf 10.2).

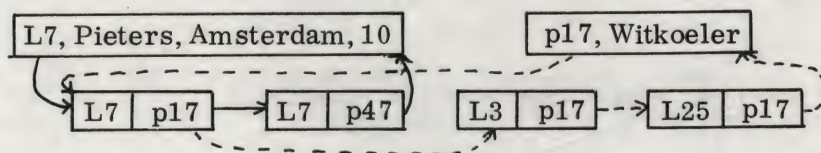


Fig. 10.10. Netwerk op occurrence-niveau

Opmerking: de term Bachman-diagram wordt alleen gebruikt voor diagrammen op objekttype-niveau.

Een gedeelte van de weergave van de record- en settypes, in een beschrijvingstaal van het netwerkmodel, volgt hieronder.

RECORD	NAME	IS	LEVERANCIER
02	LNR	PIC	XXX
02	LNAAM	PIC	X (40)
02	LWOONPL	PIC	X (30)
02	GCODE	PIC	XX

RECORD	NAME	IS	PROJEKT
02	PNR	PIC	XXX
02	PNAAM	PIC	X (30)

RECORD	NAME	IS	ARTIKEL
02	ANR	PIC	XXX
02	ANAAM	PIC	X (30)

RECORD	NAME	IS	LP
02	LNR	PIC	XXX
02	PNR	PIC	XXX
RECORD	NAME	IS	LA
02	LNR	PIC	XXX
02	PNR	PIC	XXX
02	PRIJS	PIC	9 (6)
RECORD	NAME	IS	PA
02	PNR	PIC	XXX
02	ANR	PIC	XXX
02	HOEV	PIC	9 (4)
SET	NAME	IS	LEV-PROJ
OWNER	IS	LEVERANCIER.	
MEMBER	IS	LP.	
SET	NAME	IS	PROJ-LEV
OWNER	IS	PROJECT.	
MEMBER	IS	LP.	

Ook bij de gegevensbeschrijving van dit voorbeeld is veel weggelaten, dat te maken heeft met geheugenopslag en toegangspaden. De beschrijving van de gegevensstructuur in het netwerkmodel is dan ook dikwijls niet alleen omvangrijk, maar ook onoverzichtelijk.

Normalisatie, zoals uiteengezet in de vorige paragraaf, maakt geen onderdeel uit van het netwerkmodel. Het kan overigens wel binnen het netwerkmodel worden toegepast, en dit vooral dank zij het voorschrift dat tussen owner- en member-typen slechts een 1 : n en geen n : m verband is toegestaan.

RELATIONELE MODEL

In het relationele model wordt een objekttype weergegeven door een relatie, en dit laatste dan wel te verstaan in de wiskundige zin van het woord. Wij zullen dit nader toelichten.

Bij elke attriboottype hoort een verzameling waarden, die mogelijk kunnen optreden als waarde voor dit type. Zo'n verzameling noemt men het domein van dat attriboottype. Zo behoren bij het objekttype in beschrijving 3 van paragraaf 10.2 drie attriboottypen, dus ook drie domeinen. Bij lnr kan men denken aan de letter L gecombineerd met alle geheeltallige waarden tussen 0 en 100, bij lnaam aan

alle mogelijke combinaties van maximaal 40 karakters en bij lwoonpl aan de verzameling van alle Nederlandse plaatsnamen. (In de nog volgende beschrijving zullen wij voor lwoonpl een eenvoudiger te beschrijven domein kiezen.) Op deze wijze hebben wij drie domeinen. We geven deze domeinen even kortweg aan met D1, D2 en D3. Het Cartesisch produkt van deze drie domeinen, weergegeven met $D1 \times D2 \times D3$, bestaat uit alle combinaties van drie waarden (d1, d2, d3) (in deze volgorde), waarbij d1 een element is van D1, d2 een element van D2 en d3 een element van D3. Een relatie is nu een deelverzameling van het Cartesisch produkt van de domeinen. Een element van een relatie wordt dikwijls een tuple genoemd. Een tuple is dus in feite hetzelfde als een occurrence van een objekttype en een relatie hetzelfde als de verzameling van alle aktuele occurrences van een objekttype. Een tuple van de relatie (behorende bij het objekttype) leverancier is bijv. : L7, Pieters, Amsterdam.

Binnen dit relationele model werden de normalisatiebegrippen ontwikkeld, zoals in de vorige paragraaf uiteengezet. Dit is een belangrijke stap in de ontwikkeling van gegevensstructuren, omdat hiermee volgens eenduidige afspraken de betekenis van de relevante gegevens kon worden vastgelegd. Dat dit overigens geen eenvoudige zaak blijkt te zijn voor de werkelijk optredende problemen, is reeds vermeld.

Voor het voorbeeld van de vorige paragraaf wordt nu een beschrijving gegeven volgens het relationale model.

<u>DOMAIN</u>	LNR	CHAR(3)
	LNAAM	CHAR(40)
	LWOONPL	CHAR(30)
	GCODE	CHAR(2)
	ANR	CHAR(3)
	ANAAM	CHAR(30)
	PNR	CHAR(3)
	PNAAM	CHAR(30)
	PRYS	NUM(6)
	HOEV	NUM(4)
<u>RELATION</u>	LEVERANCIER(LNR, LNAAM, LWOONPL)	
	KEY(LNR).	
	ARTIKEL(ANR, ANAAM)	
	KEY(ANR).	
	PROJECT(PNR, PNAAM)	
	KEY(PNR).	
	ART-LEV(LNR, ANR, PRYS)	
	KEY(LNR, ANR).	
	PROJ-ART(PNR, ANR, HOEV)	
	KEY(PNR, ANR).	
	PROJ-LEV(LNR, PNR)	
	KEY(LNR, PNR).	
	CODE(LWOONPL, GCODE)	
	KEY(LWOONPL).	

In een gegevensbeschrijving van het relationele model wordt, in tegenstelling tot de andere modellen, niets gezegd over toegangspaden. Zodoende resulteert (op het logische niveau) een heel doorzichtige gegevensstructuur.

De zeven tabellen in de vorige paragraaf zijn een voorbeeld van een weergave op occurrence-niveau in het relationele model.

10.4 GEBRUIK VAN DATABASE. VRAAGTALEN

Wij zagen dat voor het gebruik (en onderhoud) van een database een DML nodig is. Hierbij valt een belangrijk onderscheid te maken tussen een DML waarmee met een opdracht precies één occurrence van één objekttype kan worden benaderd, en een DML waarmee met een opdracht een aantal occurrences van een objekttype of van verschillende objekttypen kunnen worden benaderd. Deze laatste soort DML is eigenlijk de soort, die het beste 'past' bij de opzet van een database. Immers een database bevat gegevens over objekttypen en hun verbanden. Het is dan ook voor de hand liggend dat men bij het gebruik van een database ook deze verbanden wil benutten, dus ook occurrences van verschillende objekttypen. Voor deze laatste soort DML wordt veelal de naam vraagtaal (query taal) gebruikt.

Vraagtaalen staan op het ogenblik in het middelpunt van de belangstelling. Enerzijds wordt verwacht dat daar een van de zwaartepunten van de toekomstige ontwikkeling op database-gebied zal liggen; anderzijds vraagt een enigszins aan zijn doel beantwoordende vraagtaal (vooralsnog) om dure en ingewikkelde voorzieningen. Vraagtaalen worden dan ook bij de meeste database-managementsystemen als een apart produkt geleverd.

Hoe ook de toekomstige ontwikkeling op database-gebied moge zijn, het zal altijd goed zijn zich te realiseren dat nieuwe faciliteiten best de moeite waard kunnen zijn, mits het te behalen voordeel (ruimschoots) opweegt tegen de prijs, die nu eenmaal voor elke service moet worden betaald.

OPGAVEN

De opgaven 10.1 en 10.2 gaan uit van het voorbeeld in paragraaf 10.2.

- 10.1. Welke relaties in 3NF zullen ontstaan als de PRYS per artikel
 - a. onafhankelijk is van de leverancier?
 - b. afhankelijk is van artikel, leverancier en projekt?
- 10.2. a. Welke relaties in 3NF zullen ontstaan als per leverancier/artikel/projekt de hoeveelheid geleverd door een leverancier

van een artikel aan een projekt moet worden vastgelegd?
 b. Hoe kan dit in het netwerkmodel worden weergegeven?

- 10.3. Ten behoeve van de centrale (studenten)administratie van een instelling moeten per student de volgende gegevens ter beschikking staan:

registratienr

naam

adres student

jaar van aankomst

afdelingsnr

afdelingsnaam

adres afdelingsadministratie

vakcode

vaknaam

vakresultaat

naam docent

adres docent

per vak

Geef bovenstaande gegevens weer met relaties in de derde normaalvorm.

- 10.4. Bedenk een hiërarchische structuur waarin dezelfde objekttypen als in fig. 10.3 voorkomen, maar in een andere hiërarchie. Geef diagrammen op objekttype, attribuuttype en occurrence-niveau.
- 10.5. Geef de gegevens van opg. 10.3 weer met een (of meerdere) hiërarchische structuren.
- 10.6. Geef de volgende verbanden tussen AFDELING-type en WERKNEMER-type met behulp van CODASYL-sets (in een diagramvoorstelling) weer:
- (werknemer) is op dit moment werkzaam in (afdeling). Een werknemer is hooguit in één afdeling werkzaam.
 - (werknemer) was in 1970 werkzaam in (afdeling).
 - (werknemer) was tot nu toe werkzaam in (afdelingen).
 - (werknemer) is in bedrijfsraad vertegenwoordiger voor (afdelingen). Elke afdeling heeft precies één vertegenwoordiger.
 - Wat is naar aanleiding van het vergelijken van c en d op te merken. Geef van c en d ook een occurrence-diagram.
- 10.7. Ontwerp een netwerkmodel voor de gegevens van opg. 10.3.
- 10.8. Voor een handelsonderneming is een database opgezet volgens onderstaande specificatie in derde normaalvorm (sleutelitems zijn onderstreept).

DOMAIN

KN
KNM
KADR
SALDO
BADR
AFST
KRTP
ON
DAT
LT
AN
ANM
PRIJS
HOEV
VR

Betekenis

Klantnummer
Klantnaam
Klantadres (voor correspondentie)
Saldo van klant
Adres voor bezorging
Afstand tot bezorgadres (in km)
Kortingspercentage
Ordernummer
Orderdatum
Levertijd
Artikelnummer
Artikelnaam
Prijs per stuk
Bestelde hoeveelheid
Voorraad

RELATION

KLANT(KN, KNM, KADR, SALDO)
BEZORG(BADR, AFST, KN, KRTP)
ORDER(ON, DAT, BADR, LT)
ARTIKEL(AN, ANM, PRIJS, VR)
REGEL(ON, AN, HOEV)

- a. Geef kort en duidelijk commentaar op elk van de volgende uitspraken:
 1. Een klant kan meerdere bezorgadressen hebben.
 2. Meerdere klanten kunnen hetzelfde correspondentieadres hebben.
 3. Een order kan betrekking hebben op meerdere klanten.
 4. Het kortingspercentage wordt per order bepaald.
- b. Geef de database weer in het netwerkmodel met behulp van een diagram. Hierbij niet de attribuuttypen maar alleen de verschillende recordtypen vermelden.
- c. Veronderstel dat een order altijd op precies één artikel betrekking zou hebben. Hoe zou de database-beschrijving dan aangepast moeten worden om deze in derde normaalvorm te houden?

11 GEMENGDE OPGAVEN

- 11.1. In een magazijn liggen vele soorten artikelen opgeslagen. De artikelsoorten zijn verdeeld in 6 groepen. Elk van deze 6 groepen is weer onderverdeeld in 4 subgroepen. Voor iedere subgroep is het aantal artikelsoorten kleiner dan 100. In verband met de automatisering van de voorraadbeheersing willen men voor iedere artikelsoort een record maken met de artikelcode als sleutel.
- Ontwerp een artikelcode zodanig dat in deze code zowel de groepen als de subgroepen onafhankelijk van elkaar identificeerbaar zijn.
 - Ontwerp een artikelcode waarbij niet meer de eis sub a geldt maar wel de eis dat de code niet meer dan drie karakters mag bevatten.
- 11.2. Een bepaald bestand is reeds enige tijd in gebruik voor een wekelijkse toepassing met een bestandsactiviteit van 95%. Op goede gronden werd indertijd besloten dit bestand sequentieel op magneetband op te slaan. Voor een nieuwe maandelijkse toepassing met een bestandsactiviteit van 5% is het nodig dat bepaalde records worden geselecteerd op grond van de waarden van drie attributen (de sleutel behoort niet tot deze drie). Het aantal mogelijke waarden van elk van deze drie attributen is hoogstens vijf. In verband met deze nieuwe toepassing (waarnaast de reeds eerder genoemde wekelijkse toepassing van kracht blijft) worden door de systeemanalisten A, B en C de volgende voorstellen gedaan.
- A komt tot de conclusie dat op basis van de genoemde attributen een geïnverteerd bestand moet worden opgebouwd teneinde de nieuwe maandelijkse toepassing efficiënt te verwerken.
- B komt eveneens tot de conclusie dat een geïnverteerd bestand ideaal is, maar stelt daarbij voor het sequentiële bestand op band om te zetten in een direct bestand op schijf.
- C is van mening dat een geïnverteerd bestand niet nodig is, maar dat een efficiënte verwerking van de nieuwe toepassing

kan worden bereikt met een kettingorganisatie in het bestaande bestand.

- a. Wat is uw reactie op de voorstellen van A, B en C?
- b. Wat is eventueel uw eigen voorstel?

- 11.3.
- a. Is een sequentieel bestand op een adresseerbaar medium niet in feite een willekeurig toegankelijk bestand?
 - b. Waarom zal het bij een index-sequentieel bestand meestal 'niet-lonend' zijn in te voegen records op te slaan in het primaire gebied op plaatsen van verwijderde records?
 - c. Bij welke bestandsorganisaties is sprake van gereserveerde ruimte voor overflow? Is het mogelijk dat, onder bepaalde voorwaarden, voor een of meer van deze organisaties reservering van overflow-ruimte eigenlijk niet nodig is?

- 11.4. Een handelsonderneming wenst per artikel over de volgende gegevens te kunnen beschikken:

- artikelnummer
 - artikelomschrijving
 - aanwezige voorraad
 - minimum voorraad
 - leveranciers (onbepaald aantal per artikel)
per leverancier:
 - leverancierscode
 - naam, adres, woonplaats, telefoon
 - gemiddelde levertijd (onafhankelijk van artikel)
 - per aflevering in het lopende kalenderjaar:
 - afgeleverd aantal
 - faktuurbedrag
 - verkopen van de laatste 12 maanden, per maand
 - aantal verkochte exemplaren
 - verkoopbedrag.
- a. Er wordt een willekeurig toegankelijk (met functionele relatie) artikelbestand opgezet. Elk artikelrecord bevat de bovenvermelde onderdelen.
Geef voor elk van de twee volgende mutaties een aparte programmaschets.
- 1. Voor een bepaald artikel heeft een aflevering door een leverancier plaatsgevonden.
 - 2. Van een bepaalde leverancier moeten adres en woonplaats worden veranderd.
- b. Zou het aanbeveling verdienen voor ieder van de typen mutaties sub a niet met één bestand te werken, maar dit bestand te splitsen in twee of meer andere bestanden?

(Bij splitsing de recordinhoud van de nieuwe bestanden aangeven.)

c. Geef bovenstaande gegevens weer in de derde normaalvorm.

- 11.5. Gegeven is een sequentieel bestand van 100 000 records. Het verschil tussen opeenvolgende sleutelwaarden is niet constant (zie ook tabel hieronder). Het bestand ligt fysiek aaneengesloten, ongeblokt opgeslagen op een schijvenpakket. Daarbij is de mogelijkheid aanwezig een willekeurige recordplaats in dit bestand direct te benaderen. Als bijv. wordt gevraagd naar het 1500e record dan zal het systeem, zonder voorafgaand lezen van de eerste 1499 records, direct het leesmechanisme instellen op de 1500e recordpositie en dit record inlezen.

a. Kunnen we met juist genoemde mogelijkheid nu spreken van een willekeurig toegankelijk bestand? (einde vraag a)

In het bestand zijn verder enkele tientallen records via een enkelvoudige kettingstructuur met elkaar verbonden. Het adres van het eerste record van de ketting is aan het systeem bekend, evenals het totaal aantal records in de ketting. De records in de ketting liggen niet fysiek aaneengesloten. Wel geldt voor deze ketting dat elk volgend record een hogere sleutelwaarde heeft dan het voorgaande. Ter illustratie zij verwezen naar onderstaande tabel, die het begin van het bestand weergeeft. (Per record wordt hierbij alleen sleutel en eventueel verwijfsadres gegeven.)

	<u>sleutel</u>	<u>verwijsadres</u>
1e record	010085301	--
2e record	010090534	--
etc.	021016517	000007 (begin van de ketting)
	028230008	--
	028230010	--
	030060581	--
7e record	039250701	000015

De vraag is nu of het mogelijk is een record, waarvan bekend is dat het tot de ketting behoort, binair te zoeken en wel bij voorkeur binnen de ketting. Voor de oplossing van deze vraag mogen aangepaste voorzieningen worden getroffen. Drie personen A, B en C geven het volgende antwoord op deze vraag. A: geen problemen met binair zoeken binnen de ketting; dit kan immers, aangezien de record binnen de ketting sequentieel geordend liggen.

B: binair zoeken binnen de ketting is niet mogelijk; dus gegeven een recordsleutel in de ketting moet men voor binaire zoek

het hele bestand gebruiken en dit laatste is wel mogelijk.
C: binair zoeken binnen de ketting kan met behulp van een record uit een geïnverteerd bestand. Dit record zal dan alleen bestaan uit de recordsleutels van de ketting in sequentiële volgorde.

- b. 1. Geef uw oordeel over de antwoorden van A, B en C.
Hierbij moet gelet worden zowel op mogelijkheid als efficiency.
- b. 2. Wat is eventueel uw eigen oplossing?

11.6. (Zie voorbeeld in hoofdstuk 8.)

Gevraagd wordt een lijst met registratienummer, naw en vakgroepcode van de personen die

- in afdeling 22 werken en
- gehuwd zijn en
- tot vakgroep R behoren.

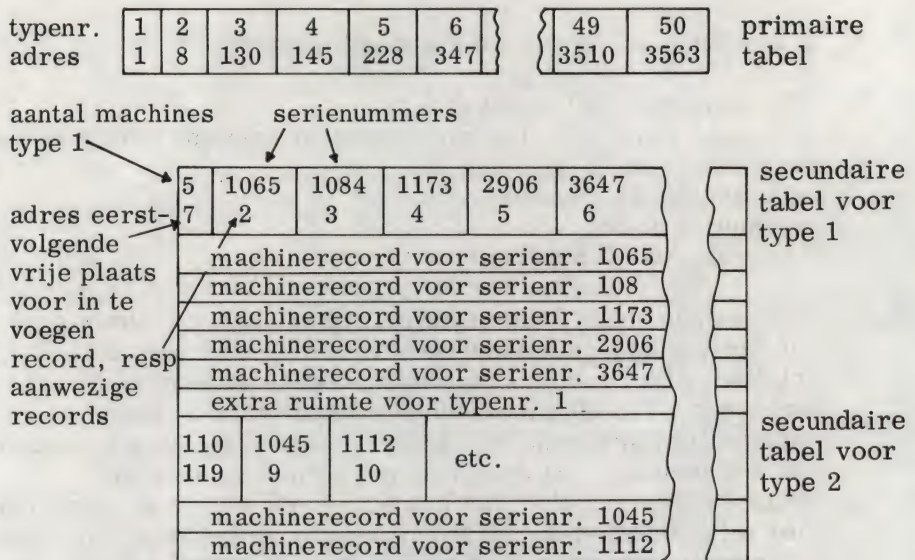
11.7. Een verhuurbedrijf van grote machines heeft de administratie van het verhuur geautomatiseerd. Het bedrijf kent 50 verschillende typen en per type minimaal 10 en maximaal 125 machines. Van elke machine worden allerlei verder niet te omschrijven gegevens opgeslagen in een vaste lengte record van 800 posities. Al deze records samen vormen het MACHINES-bestand. Elke machine heeft een uniek serienummer (niet afhankelijk van het type waartoe het behoort). Aan een nieuwe machine wordt een serienummer toegekend, dat groter is dan elk tot dusver bestaand serienummer. Een eis van het systeem is dat het MACHINES-bestand direct toegankelijk moet zijn per type. Verder moet het op serienummer binnen type zowel sequentieel als direct toegankelijk zijn. Om aan deze eisen tegemoet te komen is het bestand opgeslagen op schijf in de volgende organisatievorm.

Er is een primaire tabel voor alle typenummers en per typennummer een secundaire tabel voor alle machines behorende tot het desbetreffende type. In de primaire tabel staat per type het (relatieve) adres vermeld van de bijbehorende secundaire tabel.

Elke secundaire tabel bevat de serienummers en de (relatieve) adressen van alle bij het desbetreffende type behorende machinerecords. Bovendien staat in elke secundaire tabel vermeld het aantal machines dat tot het desbetreffende type behoort. De bij een bepaald type behorende machinerecords worden sequentieel volgens serienummer na de bijbehorende secundaire tabel opgeslagen.

De (relatieve) adressering is zodanig dat met een adres een schijfsegment wordt aangegeven. Voor elk machinerecord en

voor elk van de genoemde tabellen wordt precies een segment gebruikt. De invoerbuffer is gelijk aan de segmentgrootte. In verband met het invoegen van nieuwe records is per type ongeveer 10% extra ruimte gereserveerd. In elke secundaire tabel staat het adres vermeld van de eerstvolgende vrije plaats voor in te voegen records. Zie voor bovenbeschreven organisatievorm ook de volgende schets.



- a. Is het bij de opslag van de tabellen nodig dat
 1. in de primaire tabel de typenummers worden opgeslagen?
 2. in elke secundaire tabel de adressen van alle machinerecords (behorende bij het desbetreffende type) worden opgeslagen?
- b. Hoeveel leesopdrachten zijn (gemiddeld) nodig om een bepaald machinerecord op te halen als van dit record alleen gegeven zijn:
 1. serienummer
 2. serienummer plus typenr. waartoe dit record behoort.
- c. Zijn de secundaire tabellen nodig in verband met de genoemde eisen betreffende benaderingsmogelijkheden?
- d. Wat vindt u van de volgende uitspraak:

"Gegeven typennummer en serienummer kan het bijbehorende record worden opgezocht en ingelezen door achtereenvolgens

 - de primaire tabel in te lezen en op basis van het gegeven

typenummer via een binaire zoek het adres van de bijbehorende secundaire tabel te bepalen,

- deze secundaire tabel in te lezen en op basis van het gegeven serienummer via een binaire zoek het record-adres te bepalen,
- het gevraagde record in te lezen. "

- e. Geef een programmaschets voor het invoegen van een nieuw machinerecord, binnen een van de bestaande 50 typen; men mag hierbij veronderstellen dat
 - de extra ruimte voor nieuwe records voor ieder type altijd voldoende is,
 - het in te voegen record een sleutelwaarde heeft, die niet gelijk is aan een reeds bestaande sleutelwaarde.

11.8. (Variant op opgave 3.13.)

Het bestand STAM is direct georganiseerd met funktionele relatie, met de volgende specificaties:

- Bingrootte is 1.
- Als een bin geen record bevat, dan is de waarde van de sleutel in deze bin gelijk aan -1.
- Voor de overflow is een aparte (onbeperkt grote) ruimte beschikbaar. Het adres van de eerstvolgende vrije plaats in de overflowruimte staat in het veld 'nextfree'. De daarop volgende vrije plaats wordt verkregen met de procedure succ(nextfree).
- Synoniemen zijn door een enkelvoudige ketting met elkaar verbonden. Een nieuw in te voegen synoniem wordt als laatste record in de ketting opgenomen.

11.9. Geef op onderstaande uitspraken kort en duidelijk commentaar.

- a. Bij het invoegen van een record in een dubbele sequentiële ketting zullen hoogstens twee reeds bestaande pointers een andere waarde krijgen.
- b. Bij directe organisatie met tabelrelatie is de frequentie van reorganisatie van het gegevensbestand afhankelijk van de file turnover.
- c. Bij sequentiële verwerking van een index-sequentieel bestand is het niet nodig gebruik te maken van de index-tabellen.

11.10. Gegeven is een stambestand van 1000 artikelrecords. Elk record bevat o. a. het veld MAAT.

Het stambestand is direct georganiseerd met tabelrelatie. De bijbehorende tabel is sequentieel op oplopende sleutelwaarde, wordt afgesloten met de sleutelwaarde ∞ en kan met één leesopdracht worden ingelezen.

Bij het stambestand bestaat een geïnverteerd bestand IB. Tot IB behoort het record M46. Dit record bevat sleutelwaarde en adres van elk stamrecord met MAAT = 46.

Het vermoeden bestaat, dat record M46 niet helemaal correct is en dat met name de volgende fouten kunnen voorkomen:

- fout 1: een stamrecord met MAAT = 46 komt wel voor in het stambestand maar zijn sleutelwaarde en adres komen niet voor in record M46.
- fout 2: een stamrecord met MAAT \neq 46 komt voor in het stambestand en zijn sleutelwaarde en adres komen ook voor in record M46.
- fout 3: in record M46 komt een sleutelwaarde voor, die niet voorkomt in het stambestand.

Gevraagd een programmaschets voor de volgende opdracht. Creëer een nieuw (maar nu correct) record M46 en druk een lijst af met de gesignaleerde fouten. In deze lijst dient te worden opgenomen in geval van

fout 1: sleutelwaarde plus tekst 'hoort in IB'

fout 2: sleutelwaarde plus tekst 'hoort niet in IB'

fout 3: idem.

Met andere dan de genoemde fouten dient geen rekening te worden gehouden.

- 11.11. Een bestand, dat wordt gebruikt met een bestandsactiviteit van 100%, is sequentieel georganiseerd en is opgeslagen op magneetband. Het bestand bevat 80 000 records. Elk record heeft een vaste lengte van 300 tekens.

Gegevens betreffende de in gebruik zijnde magneetband(en):

Lengte per band : 2400 ft.

IBG : 0,4 ft.

Start + stoptijd IBG : 9 millisec.

Dichtheid : 1600 tekens per inch.

Lees/schrijfsnelheid : 100 inch/sec.

Terugspoelsnelheid : 400 inch/sec.

- a. Hoe groot moet de blokkingsfactor minimaal zijn om het bestand op één volume te kunnen opslaan?
- b. Hoeveel tijd is nodig voor het lezen (inclusief terugspoelen) van het hele bestand als de blokkingsfactor sub a wordt aangehouden?
- c. De (gemiddelde) verwerkingstijd per record is 5 millisec. Hoe groot moet de blokkingsfactor zijn, zodat de benodigde tijd voor het lezen van een blok zo dicht mogelijk in de buurt ligt van de (gemiddelde) verwerkingstijd van het blok?

11.12. Gegeven is onderstaande (sterk) vereenvoudigde beschrijving van het systeem van een touroperator voor een reis seizoen. Alle heen- resp. retourreizen worden per vliegtuig gemaakt vanuit resp. naar Amsterdam. De vluchten naar of van de verschillende bestemmingen worden op vaste dagen per week en op vaste tijden per dag uitgevoerd; bijvoorbeeld elke zaterdag en zondag, vertrektijden resp. 9.00 en 10.00 uur. Aan elke vlucht, die binnen dit systeem dus wekelijks plaatsvindt, is een uniek vluchtnummer toegekend, dat geldt voor het hele reis seizoen. Elk van deze geplande vluchten (PV) bestaat uit een of meer geplande deelvvluchten (PVD), afhankelijk van het aantal luchthavens dat per geplande vlucht zal worden aangedaan.

Per week zullen de werkelijke vluchten (WV) en werkelijke deelvvluchten (WVD) overeenkomen met de planning, m. a. w. annuleringen of extra ingelaste vluchten komen niet voor. Per boeking kan een deelnemer (DNM) voor precies één werkelijke deelvvlucht boeken. De gegevensstructuur in derde normaalvorm is onderstaand weergegeven (met weglating van hier niet relevante details).

DOMAIN

V #	vluchtnummer
DAG	dag van de week
VT	vertrektijd (in uren en minuten)
VC	vertrekcode (heen of terug)
VLT	vliegtuigtype
STPV	totaal aantal beschikbare stoelen per geplande vlucht
DV #	deelvvluchtnummer
STPVD	totaal aantal beschikbare stoelen per geplande deelvvlucht
LHC	luchthavencode
DAT	datum (jaar, maand, dag)
STB	aantal geboekte stoelen
AD	aantal deelnemers
D #	deelnemersnummer
DNAAM	deelnemersnaam
DSOM	totaal reisbedrag per deelnemer
DBET	reeds betaald bedrag per deelnemer
PV(V #, DAG, VT, VC, VLT, STPV)	
PVD(V #, DV #, STPVD, LHC)	
WV(V #, DAT, STB)	
WVD(V #, DV #, DAT, AD)	
DNM(D #, DNAAM, DSOM, DBET, V #, DV #, DAT)	

RELATION

{primary key
onderstreept}

Geef commentaar op de volgende uitspraken:

1. de identiteit van een geplande vlucht ligt alleen vast als zowel vluchtnummer als dag bekend zijn;
2. als een deelvluchnummer bekend is, is ook bekend welke luchthaven zal worden aangedaan;
3. het aantal beschikbare stoelen per geplande vlucht is niet afhankelijk van het vliegtuigtype;
4. eenzelfde persoon kan per seizoen slechts aan één werkelijke deelvluucht deelnemen.

11.13. Voor onderstaande problemen dient een korte en duidelijke oplossing geschetst te worden.

- a. Geef aan hoe een direct georganiseerd bestand met functionele relatie kan worden omgezet in een index-sequentieel georganiseerd bestand.
- b. Gegeven is een index-sequentieel georganiseerd bestand van klanten en een sequentieel georganiseerd bestand van orders van die klanten. Er is behoefte om op efficiënte wijze voor iedere klant toegang te hebben tot de orders van deze klant, waarbij men echter wil vasthouden aan de sequentiële organisatie van de orders. Welke voorzieningen moeten worden getroffen om aan genoemde behoefte tegemoet te komen? Welke consequenties hebben deze voorzieningen voor het onderhoud van de bestanden?

11.14. Een uitzendbureau ten behoeve van de verzorgende sector zendt personen uit voor bepaalde functies, in bepaalde instellingen, in bepaalde plaatsen. Het gaat om de volgende functies, instellingen en plaatsen:

functies	:	verpleegkundige A
		verpleegkundige B
		verpleegkundige C
		ziekenverzorg(st)er
		huishoudelijke dienst
instellingen	:	ziekenhuis
		verpleegtehuis
		bejaardentehuis
		dienstencentrum
plaatsen	:	Eindhoven Nuenen
		Geldrop Son
		Heeze Veldhoven
		Waalre

Een persoon kan zich bij het bureau voor precies één functie laten inschrijven, echter voor meer dan een instelling in meer dan een plaats.

Het uitzendbureau wil nu de gegevens van de ingeschreven personen vastleggen in één of meer bestanden (en eventuele

hulpbestanden). Per ingeschreven persoon moeten de volgende gegevens worden vastgelegd: registratienummer (sleutel), NAW, toewijsbare functie, toewijsbare instellingen en plaatsen, wel of niet uitgezonden. Met dit bestand (deze bestanden) moet aan de volgende eisen worden voldaan:

- I Op een willekeurig moment antwoord op de volgende vragen te kunnen geven:
 - I1 is een bepaalde persoon, waarvan het registratienummer wordt opgegeven, wel of niet uitgezonden;
 - I2 geef het aantal toewijsbare, niet-uitgezonden personen voor een bepaalde functie in een bepaalde instelling in een bepaalde plaats;
 - I3 geef een lijst met registratienummers en NAW van alle toewijsbare, niet-uitgezonden personen voor een bepaalde functie in een bepaalde instelling in een bepaalde plaats.
- II Op elk willekeurig moment moet invoegen van nieuw ingeschreven personen of veranderen van gegevens van ingeschreven personen of verwijderen van ingeschreven personen mogelijk zijn.
- III Per week een lijst te kunnen produceren met de gegevens van alle ingeschreven personen. Deze lijst moet op functie zijn gesorteerd en binnen elke functie op oplopend registratienummer.

Vraag. Welke bestandsorganisatie is het meest geschikt voor bovengenoemde persoonsgegevens. Bij uw antwoord kort en duidelijk aangeven hoe aan elk van bovengenoemde eisen op geschikte wijze wordt voldaan. Andere dan genoemde eisen spelen geen rol.

11.15. Gegeven zijn de volgende bestanden:

- LB : leveranciersbestand met de recordvelden
 - lnr : leveranciersnummer (sleutel)
 - naw : naam, adres, woonplaats
 - asp : assortimentspointer (zie verderop)
- AB : artikelenbestand met de recordvelden
 - anr : artikelnummer
 - aom : artikelomschrijving
 - vr : voorraad
 - sk : waarde 1: wel speciaal gekeurd
0: niet speciaal gekeurd
 - asp : assortimentspointer (zie verderop)
- ASB: assortimentenbestand met de recordvelden
 - lnr : leveranciersnummer combinatie(lnr, anr)
 - anr : artikelnummer is sleutel
 - pr : prijs per stuk

lvw : leveringsvoorwaarden
 lp : leverancierspointer
 ap : artikelpointer
 vaslp : pointer voor volgend
 assortimentrecord bij (zie verderop)
 zelfde leverancier
 vasap : pointer voor volgend
 assortimentrecord bij
 zelfde artikel

De assortimentsrecords met hetzelfde leveranciersnummer zijn door een ketting (met oplopend artikelnummer) met elkaar verbonden. De pointer asp in een LB-record bevat het adres van het eerste assortimentsrecord van de bijbehorende ketting. Zo zijn ook de assortimentsrecords met hetzelfde artikelnummer door een ketting (met oplopend leveranciersnummer) met elkaar verbonden, met de pointer asp in een AB-record als beginadres van de ketting. De pointer lp resp. ap in een ASB-record bevat het adres van het leveranciersrecord resp. artikelrecord met hetzelfde lnr resp. anr als in het assortimentsrecord. Hiermee hebben wij vanuit elke schakel in een ketting een verwijzing naar het bijbehorende LB-record resp. AB-record. Tenslotte bevat de pointer vaslp resp. vasap in een ASB-record het adres van het volgende ASB-record in een desbetreffende ketting, met waarde -1 aan het einde van de ketting. Het kan voorkomen dat er bij een LB-record resp. AB-record geen bijbehorende assortimentsrecords zijn. In dat geval heeft asp in het LB-record resp. asp in het AB-record de waarde -1.

LB en AB zijn sequentieel op schijf georganiseerd. ASB is direct georganiseerd met tabelrelatie. De tabel bevat per ASB-record (lnr, anr, adres in ASB-bestand) en is sequentieel met oplopend lnr en daarbinnen met oplopend anr.

Gevraagd wordt een lijst af te drukken van alle LB-records, waarvoor geldt dat de desbetreffende leverancier minstens een artikel levert en alleen artikelen levert, die speciaal gekeurd zijn.

Geef een programmaschets voor de produktie van deze lijst.

11.16. (Vervolg van opgave 11.15.)

We veronderstellen nu dat de drie genoemde bestanden reeds op het achtergrondgeheugen zijn opgeslagen, maar dat de kettingen nog niet zijn aangelegd, dus de pointers nog niet de juiste waarden bevatten.

Geef een programmaschets voor het aanleggen van de kettingen. Daarbij mag men o. a. gebruik maken van de volgende

procedures:

- sort (<tabel>, ascending <keylist>) voor het eventueel sorteren van het tabelbestand.

Zo zal sort(tab, ascending anr, lnr) de tabel met naam tab sorteren naar oplopend anr en daarbinnen naar oplopend lnr.

- readnextplus(<filename>, <recordname>, <adres>); levert in <adres> ook het (relatieve) adres van het juist gelezen record in een sequentieel bestand.

Zo zal na uitvoering van readnextplus(mf, mr, madres) in madres het (relatieve) adres van het juist gelezen record in mf staan.

- writenextplus(<filename>, <recordname>, <adres>): schrijft een record naar het (relatieve) <adres> in een sequentieel bestand.

APPENDIX I

EIGENSCHAPPEN VAN ACHTERGROND-GEHEUGENS

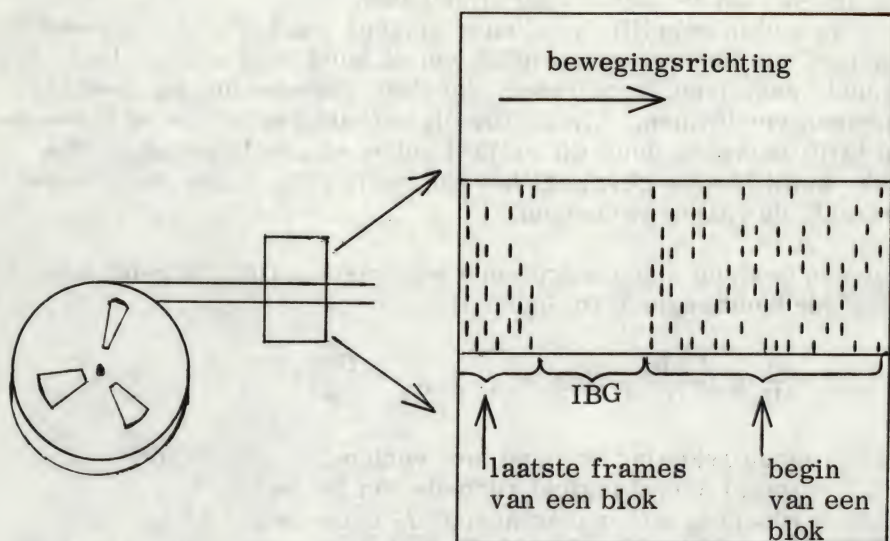
1. MAGNEETBAND

Over de breedte van een magneetband worden karakters met 8 bits (vroeger ook wel 6 bits) per karakter weergegeven. Zo'n groepje bits per karakter wordt een *frame* genoemd. Aan ieder frame wordt ook nog een controlebit toegevoegd om te kunnen nagaan (pariteitscontrole) of fysisch geen fouten zijn ontstaan bij het schrijven of lezen van karakters. Het is goed hierbij op te merken dat deze controle zich geheel aan het oog van gebruikers onttrekt. Wel zal de gebruiker bij offertes van leveranciers worden geconfronteerd met benamingen als 9-sporen- (en 7-sporen-) banden.

De *dichtheid*, waarmee een band wordt beschreven is tegenwoordig als regel 800 of 1600 karakters per inch (1 inch = 2,54 cm), genoteerd met 800 bpi (bits per inch). Tussen twee 'blokken' met karakters bevindt zich een onbeschreven gedeelte, de zogeheten *interblock gap* (IBG), van ongeveer 0,6 inch, die wel enige 'ruis' kan bevatten. Andere benamingen zijn: *blokhiat* en, minder juist, *interrecord gap*. Voor een illustratie van het tot nu toe behandelde zij verwezen naar figuur 1.

De lengte van de IBG is niet exact op te geven, omdat het fysisch niet mogelijk is het schrijven van een blok precies op een bepaalde plaats te doen beginnen. Het is zelfs mogelijk de IBG programmatisch (uiteraard via systeem-programmatuur en niet via applicatie-programmatuur) te vergroten. Dit zal met name nodig zijn als enige keren vergeefs geprobeerd is een blok correct op band te schrijven. Dit kan gebeuren als een stukje band beschadigd is. In zo'n geval kan dan het blok worden weggeschreven voorbij de kennelijk beschadigde plaats. Het moet dan ook als een ernstige fout worden aangemerkt als men bij banden probeert een gelezen blok na mutatie weer op de oude plaats terug te schrijven.

Het bijwerken van een bandbestand impliceert zodoende altijd een herschrijven op een nieuwe band! De oude band kan zo nodig nog voor bestandsreconstructie worden bewaard. Dit heeft aanleiding gegeven tot het zgn. grootvader-vader-zoon-systeem. In



Figuur 1. 'Vergrote' weergave van een detail van een magneetband.

dit systeem mag een band pas worden overschreven met nieuwe informatie als hij de status van 'grootvader' heeft bereikt, m. a. w. als een 'vader' en 'zoon' band zijn gegenereerd. Op deze manier heeft men, in noodgevallen, enkele generaties van een bestand (met in de loop der tijd aangebrachte mutaties) tot zijn beschikking. Het betekent uiteraard een investering van meerdere banden per bestand, doch dit is met het oog op de gewenste veiligheid zeer zeker verantwoord.

Alvorens nu verder te gaan met de hoofdzaken, eerst nog enige toelichting betreffende controle bij schrijven of lezen van een band. Na elke schrijf- of leesopdracht op band, betrekking hebbend op een heel blok, moeten de volgende controles worden uitgevoerd:

- a. controle op fouten;
- b. controle op einde band (bij schrijven) of einde bestand (bij lezen).

We zullen ons eerst beperken tot de controle op fouten. Vele magneetbandstations geven de mogelijkheid tot een zgn. read-after-write (ook wel: read-on-write). Dit betekent dat het schrijven 'op de voet' wordt gevolgd door een leescontrole. Daarvandaan ook de naam echo-check. Op deze manier kunnen schrijffouten onmiddellijk na het schrijven van een blok worden ontdekt. Is deze echo-check niet mogelijk, dan moet foutencontrole apart worden geprogrammeerd bijvoorbeeld met behulp van een rewind tot de vorige IBG, gevolgd

door lezen. Dit is echter zeer tijdrovend.

Nu zullen schrijffouten, en overigens ook leesfouten, meestal te wijten zijn aan verontreiniging van de band door stofdeeltjes. Bij herhaald schrijven, c. q. lezen, zal deze verontreiniging over het algemeen verdwijnen. Als echter bij herhaald schrijven of lezen de fout blijft bestaan, duidt dit vrijwel zeker op een beschadigd stuk band. In dit laatste geval zal het dan nodig zijn, zoals reeds boven vermeld, de IBG te verlengen.

Voor een bestand met records met een vaste structuur geldt voor de benodigde bandlengte L (in inches):

$$L = \frac{A}{B} \times \left(\frac{C \times B}{D} + \text{IBG} \right) = A \times \left(\frac{C}{D} + \frac{\text{IBG}}{B} \right)$$

waarbij (geen rekening houdend met verlenging van de IBG):

- A = totaal aantal logical records van het bestand;
- B = blocking factor (aantal logical records per blok);
- IBG = interblock gap (in inches; veel voorkomende maat is ongeveer 0.6");
- D = beschrijvingsdichtheid (in karakters/inch; bijv. 200, 556, 800, 1600 of 3200);
- C = aantal karakters/(logical) record.

Hieruit volgt direct dat het met de gebruikelijke waarden van D en IBG weinig zin heeft een blok lengte (CxB) te kiezen, die kleiner is dan 500 à 1000 karakters. Immers, de band zou bij kleinere blok lengte grotendeels onbeschreven zijn, d. w. z. hoofdzakelijk uit IBG's bestaan.

Factoren, die anderzijds pleiten tegen een excessief grote waarde van de blok lengte zijn:

- Tenminste dezelfde bufferruimte van CxB karakters moet in het interne geheugen beschikbaar zijn om informatie naar/vanuit een magneetband te transporteren. Zo'n bufferruimte is uiteraard qua lengte aan duidelijke begrenzingen gebonden.
- De kans om per blok een beschadigd stukje magneetband te ontmoeten neemt uiteraard toe met de blok lengte, zodat met toename van de blok lengte ook de gemiddelde waarde van de IBG zal toenemen. Blokken zullen dan ook zelden meer dan 5000 à 10 000 karakterposities bevatten.

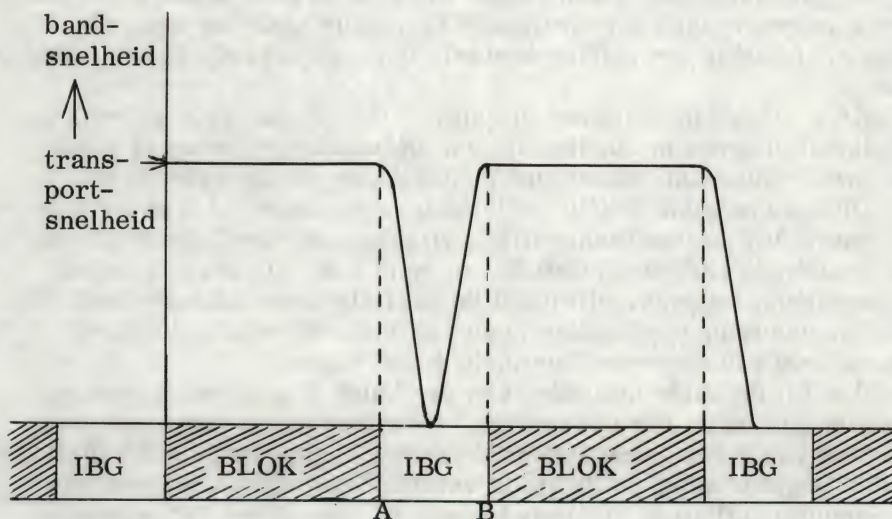
Bij betrekkelijk kleine bestanden wordt van een magneetband vaak niet veel meer dan de eerste (van de ca 700) meters gebruikt. Het 'kortwieken' van versleten beginstukken van een band was zodoende gebruikelijk. Het gebruik van trommels of schijven is voor deze kleine bestanden te prefereren, ook al omdat deze niet, zoals bij magneetbanden, door de machine-operateur klaargezet hoeven te worden. Door de ontwikkeling van schijven beperkt de rol van mag-

neetbanden zich meer tot de opslag van grote hoeveelheden (historische) gegevens.

Naast de records met vaste structuur komt men ook de volgende structuren tegen:

- ongeblokte variabele lengterecords, waarbij in het begin van ieder record de recordlengte moet worden vastgelegd;
- geblokte variabele lengterecords, waarbij in het begin van ieder blok de blok lengte wordt vastgelegd en in het begin van ieder logical record de logical recordlengte;
- 'ongedefinieerde' lengterecords, waarbij geen blok- en recordlengtes zijn vastgelegd, zodat uiteraard geen blocking mogelijk is en waarbij de controle mogelijkheden gering zijn.

Het lezen en schrijven van een band gebeurt met behulp van een lees-schrijf-kop, waaronder de te lezen/schrijven band zich voortbeweegt. Zoals we boven reeds zagen, gebeurt het lezen en schrijven van een band per blok apart. Na het transport van een blok zal de band dan ook tot stilstand moeten kunnen komen, om vervolgens voor het transport van het volgende blok weer te starten en op volle snelheid te komen. Tijdens stilstand van de band zal de lees-schrijf-kop zich boven een IBG bevinden. Een en ander wordt toegelicht in figuur 2.



Figuur 2. Bandsnelheid bij doorgang van blok en IBG.

Uit bovenstaande is wel duidelijk dat de lees/schrijftijd per blok wordt bepaald door verschillende factoren, die kenmerkend zijn voor een bepaald type magneetband. Deze factoren zijn:

- a. De start- en stoptijd (van 10 tot 40 millise.); hieronder verstaan

we de tijd die nodig is om een IBG te overbruggen, exclusief de rusttijd (uiteraard in de veronderstelling dat bij elke IBG wordt gestopt). In feite gaat het hier dus om de tijd die, exclusief de rusttijd, nodig zal zijn om het stukje AB in figuur 2 de lees-schrijf-kop te laten passeren.

- b. De (band)transportsnelheid, d.w.z. de snelheid die vereist is voor lezen of schrijven. Deze snelheid (van 37,5 tot 250 inches/sec) en de dichtheid bepalen samen de eigenlijke lees/schrijfsnelheid (van 15 000 tot 400 000 karakters/sec).
- c. De lengte van het blok (in karakters).

Uit het voorgaande is duidelijk dat een hoge leessnelheid (bijv. van 100 karakters/milliseconde) bij een kleine blok lengte (minder dan 500 karakters) 'weinig zoden aan de dijk' zet. (Vergelijk het verschijnsel: veel sneller rijden over relatief korte stukken doet de gemiddelde snelheid nauwelijks toenemen.)

Betreffende het lezen van een band moet voor een merkwaardige misvatting worden gewaarschuwd. Door de gebruikte terminologie ontstaat nl. nogal eens het idee dat 'ter plekke' van de leeskop reeds de te lezen gegevens zouden worden 'herkend' en dus bijv. zouden kunnen worden vergeleken met andere gegevens, enz. Deze herkenning, vergelijking, enz., kan echter uitsluitend plaatsvinden door het centrale rekenorgaan na voorafgaand transport naar het centrale geheugen. Leeskoppen zonder centrale organen zijn als ogen zonder hersenen.

Als een band beschreven of gelezen is, zal het voor gebruik in een volgend programma nodig zijn dat de band teruggespoeld wordt (rewind). Meestal gebeurt dit onmiddellijk na het gebruik van de band. Dit betekent dat de tijd nodig voor het terugspoelen moet worden opgeteld bij de lees/schrijftijd, wil men een idee hebben van de totaal benodigde tijd voor gebruik van een band. Dit terugspoelen gaat overigens ongeveer vijf maal zo snel als lezen of schrijven. Dus tegenover een transportsnelheid van 100 inches/sec zal een terugspoelsnelheid van ongeveer 500 inches/sec staan.

Vlak bij de beide uiteinden van een band is gewoonlijk een aluminium laagje op de band bevestigd. Hierdoor is het mogelijk (met behulp van een fotocel) het laadpunt aan het begin van de band (BOT of BOR: 'begin of tape' of 'begin of reel') of het waarschuwingsteken 'einde-band' (EOT of EOR) waar te nemen. Zo zal het terugspoelen worden gestopt bij waarneming van het laadpunt, tenzij de band van het station moet worden verwijderd. In dit laatste geval zal uiteraard worden teruggespoeld tot het begin van de band. Bij het schrijven zal waarneming van de EOR een signaal (naar de centrale eenheid) tot gevolg hebben om te waarschuwen dat de band (bijna) aan zijn eind is.

In een rekencentrum zal meestal een grote hoeveelheid banden beschikbaar zijn. Het spreekt vanzelf dat daarvoor enige administra-

tie nodig is en met name ook automatisch te verwerken administratie. Deze laatste is opgenomen in de zgn. labels, die als aparte label-records aan het bestand worden toegevoegd. Aan het begin van een bestand zullen labels opgenomen zijn met bijvoorbeeld de volgende informatie:

- fysiek bandnummer (dat ook op de bandspoel is aangebracht);
- identificatiegegevens over het programma, dat het beschrijven van de band verzorgde;
- identificatiegegevens betreffende het bestand;
- uitveegdatum (scratch-date), d.w.z. de datum waarop de band op zijn vroegst voor (over)schrijven beschikbaar is;
- maximale lengte; enz.

Behalve beginlabels (header-labels) zijn er ook eindlabels (trailer-labels). In deze eindlabels kan (gedeeltelijk) de informatie van de beginlabels staan plus bijvoorbeeld het aantal blokken van het bestand, informatie over de toestand van de band (aantal fouten), enz. Met name zal ook op een of andere wijze het einde van een bestand 'genoteerd' moeten worden (end of file: EOF, wel te onderscheiden van end of tape: EOT).

Header- en trailer-labels kunnen worden onderscheiden in:

- system-labels (aan te brengen door de standaard programmatuur),
- user-labels (aan te brengen door de gebruiker).

Wordt een band gelezen, dan zullen eerst de header-labels worden gelezen. Hiermee wordt dan o. a. gecontroleerd of de juiste band aangesloten is. Bij schrijven kan bijvoorbeeld worden gecontroleerd of de band vrij is voor het beschrijven. Bij het lezen zal het EOF aangeven dat het hele bestand is gelezen.

Voor de volledigheid zij tenslotte opgemerkt dat bij sommige fabrikaten ook teruglezen (backward reading) van een band mogelijk is.

2. MAGNEETTROMMEL

Magneettrommels waren eerder in gebruik dan magneetband (en zelfs kernengeheugens), maar hebben zich met ups en downs nog altijd gehandhaafd, vooral als buffergeheugen tussen hoofdgeheugen en periferie-apparatuur en/of magneetbanden.

Het principe van een trommelgeheugen is dat informatie op een 'spoor' op het magnetiseerbare cilindervlak van een met constante snelheid draaiende trommel geschreven/gelezen wordt met behulp van een vaste schrijf-lees-kop per spoor. De constructie is zodanig dat:

- in tegenstelling tot magneetband de positionering van het lees/schrijfmechanisme zo nauwkeurig is dat informatie (eventueel na verandering) wel precies op de oude plaats teruggeschreven kan worden;

- gemiddeld na een halve omwentelingstijd (dit is afhankelijk van het type na 5-20 ms) een willekeurig blok informatie uitgelezen kan worden (het selecteren van de gewenste leeskop en het overbrengen van informatie naar het kerngeheugen zijn processen, die zich met snelheden van 0.3 tot 1.2 M karakters/sec afspelen);
- in tegenstelling tot vele nog te bespreken schijfgeheugens de trommels niet verwisselbaar zijn.

In verband met mechanische sterkte- en balanceringsproblemen kunnen trommelgeheugens niet èn snel èn groot gemaakt worden. Langzame grote trommelgeheugens (capaciteit 10 M karakters, halve omwentelingstijd 20 ms) worden verdrongen door schijfgeheugens, snelle kleine trommelgeheugens (capaciteit 1 M karakters, halve omwentelingstijd 5 ms) door kerngeheugens (met lees/schrijftijden van de orde van 0.01 ms!), zodat het de vraag is of trommelgeheugens ondanks hun hoge betrouwbaarheid nog een lange toekomst hebben. De kosten per bit bedragen afhankelijk van de snelheid 0,5 tot 1,5 cent (inclusief de besturingseenheid).

Opgemerkt moet worden dat in sommige trommelgeheugens lees/schrijfkoppen niet vast opgesteld zijn, maar bevestigd op een gemeenschappelijke arm die evenwijdig met het cilinderoppervlak over een aantal spoorposities bewogen kan worden. Bij de halve omwentelingstijd van de trommel komt dan echter ook nog de positioneringstijd van de arm, die afhankelijk van de afstand waarover de arm zich moet verplaatsen van de orde van tientallen ms is. De som van halve omwentelingstijd en positioneringstijd is 40-100 ms, de capaciteit van 20 tot 150 M karakters, de kosten per bit zijn een factor 10 lager dan bij trommels met vaste koppen, de lees-schrijfsnelheden zijn ongeveer 0.1 tot 0.2 M karakters/sec.

Net zoals op magneetbanden (en de nog te bespreken magneetschijfgeheugens) kan de vastlegging van records op een spoor met verschillende 'formats' gebeuren. Ook hier kent men de begrippen blok-gap (die echter heel klein is omdat die niet meer een functie voor inloop en uitloop van de bandbeweging heeft), bloklength, enz. Bovendien is op een spoor ook besturingsinformatie (zoals nummer van een spoor, begin van een spoor, aantal records op een spoor, enz.) vastgelegd. Deze laatste informatie is echter veelal niet toegankelijk voor de gebruiker, maar wel voor het operating system voor controledoelinden.

De bloklength moet bij voorkeur een geheel aantal keren 'passen' op de maximaal beschikbare spoorcapaciteit. Leveranciers verstrekken als regel tabellen waaruit het aantal blokken/spoor en de eigenlijke leestijd afgelezen kunnen worden als functie van de bloklength.

De belangrijkste functie van snelle trommels is de opslag van programma's (bij gebruik van virtuele geheugens en bij time-sharing)

en harde software die snel beschikbaar moeten zijn. De langzamere trommels worden vooral voor massale gegevensopslag gebruikt.

3. MAGNEETSCHIJVEN

Magneetschijfgeheugens zijn na ongeveer 1965 in toenemende mate in gebruik genomen om tegemoet te komen aan de wens naar goedkope, willekeurig toegankelijke secundaire geheugensystemen. In principe bestaan zij uit een enkele schijf (cartridge) of een pakket (disk-pack) van 6-10 op een gemeenschappelijke as draaiende schijven waarvan de twee oppervlakken magnetiseerbaar zijn. Informatie wordt op (200-500) concentrische sporen (niet spiraalvormig zoals op een grammofoonplaat!) geschreven met behulp van een lees-schrijf-kop voor ieder oppervlak. Deze koppen zijn op een arm ('kam') bevestigd, die evenwijdig met de schijfoppervlakken heen en weer kan bewegen, zodat de koppen boven ieder spoor gebracht kunnen worden (zie tekening aan het einde van deze appendix). De sporen op de oppervlakken behorende bij eenzelfde kampositie vormen een zgn. 'cylinder'. De capaciteit per spoor is niet afhankelijk van het cylindernummer (zodat de binnenste sporen met een hogere dichtheid geschreven moeten worden). De constructie is zodanig dat:

- zoals bij magneettrommels de positionering van het lees/schrijf-mechanisme zo nauwkeurig is, dat informatie (eventueel na verandering weer precies op de oude plaats teruggeschreven kan worden;
- het uitlezen van een willekeurig blok informatie niet alleen als bij trommels een halve omwentelingstijd (ongeveer 10-20 ms) vraagt (search time of latency time), maar ook de tijd nodig om de kam te verplaatsen als de leeskop niet toevallig boven het goede spoor staat (seek time). De gemiddelde armpositioneringstijd ligt tussen de 30 en 100 ms en is dus als regel de belangrijkste factor in de totale lees/schrijftijd (net als bij trommels gaan het elektronisch selecteren van de leeskop en het overbrengen van karakters met snelheden van 0.1 tot 0.8 M karakters/sec). Het kan dan ook zeer tijdbesparend werken als opeenvolgende benodigde blokken in dezelfde cylinder staan;
- de kleinere schijfpakketten (tot een capaciteit van ca 30 miljoen karakters; kosten per bit ca 0,01 tot 0,04 cent) veelal uitwisselbaar zijn (en daarvoor in veel opzichten gestandaardiseerd); de grotere met een capaciteit tot een paar honderd miljoen karakters zijn vast opgesteld. De laatste hebben veelal een vaste kop per spoor, zodat de armpositioneringstijd vervalt en de karaktertransportsnelheid iets hoger is (0.3 tot 1.5 M kar/sec); de kosten per bit zijn ca 0,1 tot 0,6 cent (vergelijk deze gegevens met die van trommels!).

De uitwisselbaarheid van schijfpakketten heeft vele voordelen:

- de secundaire geheugencapaciteit kan (met het bezwaar van manuele handelingen) worden uitgebreid zonder direct nieuwe schijfgeheugen-

kasten aan te schaffen;

- bij het uitvallen van een schijfgeheugenkast kan het pakket overgezet worden op een andere kast en het werk kan worden voortgezet;
- bij het uitvallen van een hele installatie kan in principe het werk voortgezet worden op een soortgelijke installatie (in de praktijk treden hierbij toch wel vaak organisatorische problemen op!).

Voor bestanden op schijf zullen, evenals voor bandbestanden, labels nodig zijn. Hiervoor zij verwezen naar hetgeen hierover bij de behandeling van de magneetband werd gezegd, tenminste wat betreft header-labels. Trailer-labels worden nl. op schijf (meestal) niet gebruikt.

Schijfgeheugens zullen misschien op den duur verdrongen worden door 'solid-state' of door optische (laserstraal) geheugensystemen, wanneer deze door het experimentele stadium heen zijn. Tot ca 1985 wordt echter verwacht dat door toenemende capaciteit per pakket (30 M karakters in 1967, 300 M karakters nu) de opslagkosten per bit verder zullen dalen, zodat schijven in toenemende mate trommels en banden zullen verdringen. Dit moet uiteraard niet gedachteloos gebeuren. Men bedenke immers dat met name bij een voldoende grote bloklength (enkele duizenden karakters) de totale leestijd per blok voor magneetschijf en magneetband dikwijls vrijwel gelijk kan zijn. Voor de sequentiële verwerking van een high-activity bestand opgebouwd uit zulke records biedt een schijfgeheugen dan geen voordelen boven een magneetband, die bovendien een veel goedkopere informatiedrager is. Pas wanneer het bestand een low-activity heeft, of de verwerking niet-sequentieel geschiedt, biedt een schijfgeheugen voordelen.

4. MAGNEETSTRIPGEHEUGENS

Deze geheugens vormen wat gebruikskarakteristiek betreft enige gelijkenis met schijfgeheugens, al zijn ze nog een factor 10 langzamer. Informatie wordt vastgelegd op stukjes magneetband, waarvan een aantal op een plastic drager is geplakt. Deze dragers zitten in een bakje. Hieruit wordt langs elektromechanische weg de gewenste drager geselecteerd, de drager wordt op een draaiend trommeltje opgespannen, waarna een beweegbare of vaste lees-schrijf-kop als bij een trommel de gewenste informatie afleest. De toegangstijd van ongeveer $\frac{1}{2}$ s maakt dat deze stripgeheugens alleen voor sequentiële informatieverwerking geschikt zijn. Bovendien zijn ze tot dusver meestal niet erg betrouwbaar gebleken, zodat de toekomst van stripgeheugens ondanks lage kosten per bit erg onzeker is.

5. INFORMATIETRANSPORT TUSSEN SECUNDAIRE INFORMATIE-DRAGERS EN HOOFDGEHEUGEN

De schrijver van een proceduretaalprogramma verwacht met voor een bepaalde taal karakteristieke opdrachten - die we voortaan READ en WRITE zullen noemen - een logisch record te lezen of weg te schrijven. Anderzijds bewerkstelligen hardware opdrachten - die wij GET en PUT zullen noemen - de uitwisseling van informatie tussen hoofdgeheugen en achtergrondgeheugen. Wat is nu de samenhang tussen deze opdrachten?

Deze samenhang wordt tot stand gebracht door het operating system dat bijvoorbeeld met een GET opdracht fysiek informatie-transport van een secundair geheugen naar een 'bufferruimte' in het hoofdgeheugen veroorzaakt. Het eerste logische record vervat in het blok in de bufferruimte wordt dan met de eerste READ opdracht 'toegankelijk' gemaakt voor verdere behandeling (of dit nu gebeurt ter plaatse of dat het logical record weer verhuisd wordt naar een andere plaats is voor de gebruiker verder niet van belang). Met iedere volgende READ opdracht wordt telkens het volgende logische record toegankelijk gemaakt tot het laatste in de buffer toe. Alvo-rens de volgende READ opdracht te gehoorzamen moet het operating system eerst een GET opdracht uitvoeren, waarmee de buffer met het volgende blok gevuld wordt, enz.

Om te voorkomen dat het verwerkingsprogramma moet wachten op de uitvoering van de GET opdracht, werkt men soms met twee, of drie buffers, die door het operating system alvast 'vooruit' gevuld worden zodra een bufferruimte vrijkomt. Dit heeft uiteraard alleen maar zin wanneer de bewerkingstijd van de logical records in een buffer gemiddeld niet korter is dan de vultijd van een buffer omdat de informatiebewerking anders 'inloopt' op het informatie-transport. Is omgekeerd de bewerkingstijd veel groter dan de buffervultijd dan is een tweede buffer ook niet zinvol, omdat dan een stuk kernengeheugen lange tijd 'ongebruikt' blijft.

Uit deze korte karakterisering volgt al dat het geheel vrij gecompliceerd is. Gelukkig hoeft de gebruiker dit niet meer zelf te programmeren. Een onderdeel van het operating system is tegenwoordig het data management system, waarin deze zaken voorgeprogrammeerd zijn. De gebruiker hoeft nog maar in o. a. de 'karwei besturingstaal' ('job control language') de parameters voor zijn situatie op te geven, zoals recordlengte, aantal buffers, enz.

Het zal echter duidelijk zijn dat het vaststellen van de optimale blok- en bufferlengte en het aantal buffers afhankelijk is van factoren als:

- hardware eigenschappen (toegangs- en informatietransporttijden);
- software eigenschappen (hoe lang duurt de verwerking van READ en GET opdrachten, gezien alle controles die onzichtbaar voor de gebruiker daarbij uitgevoerd worden door het operating system);
- probleemeigenschappen die niet alleen bepalen hoeveel ruimte in

het hoofdgeheugen nog beschikbaar is voor buffers, maar ook hoe lang de verwerking van een logical record duurt.

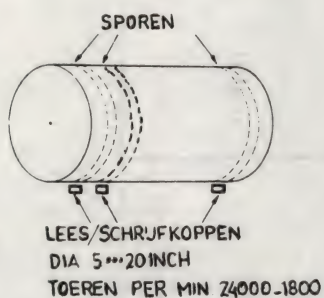
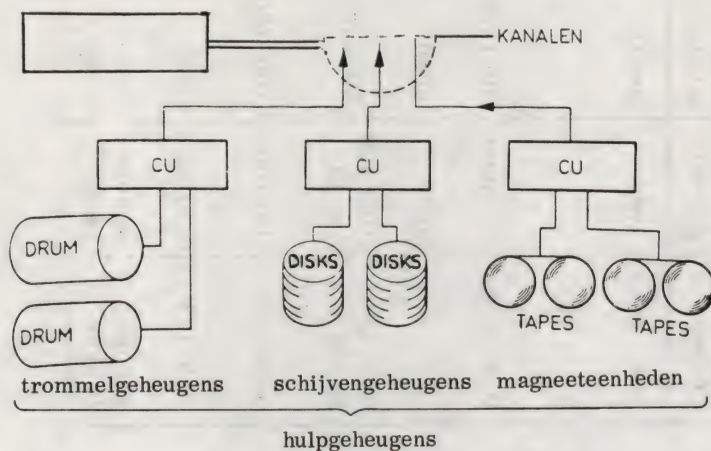
Een voor alle toepassingen geldig wiskundig model is hiervoor nauwelijks op te zetten, zodat van geval tot geval dit probleem bestudeerd moet worden. Dat het niet triviaal is, kan hieruit blijken dat de verwerkingstijd van een slecht opgezet systeem factoren drie tot vijf langer kan zijn dan bij een goede **opzet**. Bij een eerste verkenning van een probleem hanteert men meestal vuistregels van het type

- zorg ervoor dat een instructiereeks niet meer dan 5000 tot 20000 woorden in het geheugen beslaat (dit correspondeert met ongeveer 500 tot 1000 statements in een proceduretaal, die voor een programmeur nog goed te overzien zijn);
- zorg ervoor dat bloklengtes tussen 500 en 5000 karakters zijn, daar dit een redelijk compromis is t. a. v. leestijd en gebruik van band en schijf;
- voorkom, indien enigszins mogelijk, het werken met 'multivolume' files (meer dan één drager nodig voor één bestand) en 'multifile' magneetbandbestanden (meer dan één bestand op één magneetband) om verwerkingstijd te besparen.

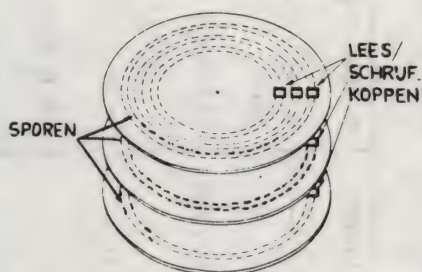
OVERZICHT VAN ENKELE INFORMATIEDRAGERS

	ponsband	ponskaart	magneetband	magneet- trommel	schijf- pakket	magneet- strip
betrouwbaarheid	goed	goed	goed	zeer goed	goed	gering
kosten inf.drager	$\sim f$ 3,-/rol	$\sim \frac{1}{2}$ ct/kaart	$\sim f$ 100,-/band			
capac. inf.drager (in karakters)	0.1 M	80 kar	4-25 M	0.1-10 M	5-300 M	50-1000 M
opbergmogelijkheid	gering	goed	zeer goed		goed	goed
leessnelheid (in duizendtallen)	0.5-2 kar/s	0.5-2 cards/m	15-400 kar/s	300-1200 kar/s	100-800 kar/s	50 kar/s
random toegangstijd (ms)				5-20	10-100	200-700
kosten per kar (ct)		0.005	~ 0.001	0.5-1.5	0.01-0.6	~ 0.5
max. physical record	$\sim \infty$	80 kar	$\sim \infty$	spoor- lengte	spoor- lengte	spoor- lengte

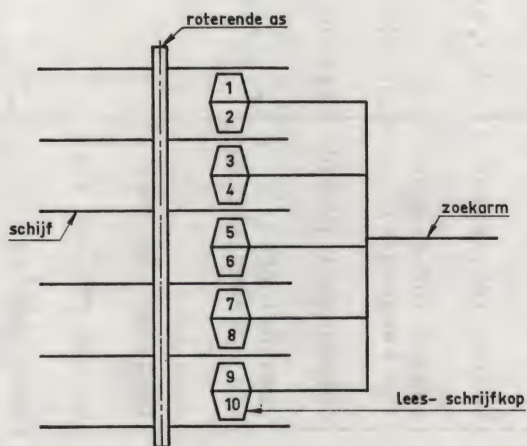
PLAATS VAN DE HULPGEHEUGENS IN HET COMPUTERSYSTEEM



Trommelgeheugen met individuele koppen per spoor



Schijfgeheugen met individuele koppen per spoor



Schijvenpakket met 'kam' van lees-schrijfkoppen.

APPENDIX II

TOELICHTING OP DE TAAL GEBRUIKT IN PROGRAMMASCHETSEN

1. Voor de toekennings-, voorwaardelijke en herhalingsopdracht worden de volgende constructies gebruikt:

toekenning : <variabele> := <expressie>;

Van de mogelijk voorkomende expressies noemen we hier alleen die met de zogeheten mutatie operatie:

<variabele> \oplus <variabele>

Met \oplus wordt aangegeven dat de eerste variabele wordt gemuteerd met behulp van de tweede, bijv.

oudrec := oudrec \oplus mutrec

voorwaardelijk : if <condition>
 then <statements>
 fi;
 of
 if <condition>
 then <statements>
 else <statements>
 fi;

herhaling : while <condition>
 do <statements>
 od;

2. DEFINITIE VAN RECORD- EN FILE-TYPEN

Een record-type wordt syntactisch als volgt gedefinieerd:

record <recordname>[(<fieldlist>)]

met

<fieldlist> ::= <fieldname> {,<fieldname>}.

Opmerking:	[]	niet verplicht
	{	}	0 of 1 of meer malen

Bij elk recordtype kunnen 1 of meer filetypeen worden gedefinieerd als volgt:

file <filename> of <recordname>

Een groep type-definities wordt omsloten door de symbolen type en endtype.

VOORBEELD:

```

type record persoon(regnr, naam, adres, wpl),
                artikel(artnr, naam, hoev),
                transactie(artnr, transcode, gegevens)
    file         personen of persoon,
                artikelen of artikel,
                transacties of transactie
endtype;

```

Bij de definitie van records wordt volstaan met het noemen van de veld-namen. We gaan hierbij van de veronderstelling uit dat de typen van de velden vastliggen en aan het systeem bekend zijn.

3. DECLARATIE VAN RECORD- EN FILEVARIABLEN

Gebieden voor files (op het achtergrondgeheugen) en records (in het werkgeheugen) komen beschikbaar voor een programma via declaraties van overeenkomstige variabelen binnen het desbetreffende programma of in een omvattende omgeving van dit programma. De daarvoor gebruikte syntax is:

```

<variable name list> : <typename>
<variable name list> ::= <variablename>{,<variablename>}
<variablename> ::= <filevariablename>|<recordvariablename>
<typename> ::= <recordname>|<filename>

```

Een groep declaraties wordt ingeleid met het symbool var en afgesloten met het symbool endvar.

Voor files zullen we aannemen dat ze altijd in een het programma omvattende omgeving zijn gedeclareerd, om te voorkomen dat ze buiten het programma niet (vooraf en/of achteraf) beschikbaar zouden zijn.

In deze omvattende omgeving wordt met de procedure

```
ready(<filenamelist>)
```


voorafgaande aan het programma bewerkstelligd, dat de inputfiles gereed liggen met de gewenste records.

De procedure

```
save(<filenamelist>),
```

zal (na het programma) zorgen dat de outputfiles beschikbaar blijven.

Declaratie van recordgebieden in het werkgeheugen dient uiteraard te geschieden binnen het programma zelf. In het programma wordt aan een recordveld gerefereerd met de syntax:

```
<recordvariablename>.<fieldname>
```

4. LEES- EN SCHRIJFPROCEDURES

- Voor sequentiële verwerking zijn de volgende procedures beschikbaar:

```
lezen      : readnext (<filevariablename>,<recordvariablename>)
schrijven  : writenext(<filevariablename>,<recordvariablename>)
```

Als *f* de desbetreffende *filevariablename* en *r* de desbetreffende *recordvariablename* is en bovendien *sl* de naam van het sleutelveld in *r*, dan is het resultaat van de procedure *readnext* resp. *writenext* dat

van resp. naar het aan de beurt zijnde adres van het sequentiële bestand *f* een record wordt gekopieerd

naar resp. van het recordgebied *r* in het werkgeheugen.

Bovendien zal *readnext* bij constatering van de end-of-file toestand aan *r.sl* de waarde ∞ (symbolische weergave van 'highest value') toekennen. Hierbij moet worden opgemerkt dat de end-of-file toestand niet wordt geconstateerd bij het lezen van het laatste record van het bestand maar bij de eerste daarop volgende leesopdracht.

- Voor niet-sequentiële verwerking zijn beschikbaar:

```
lezen      : read (<filevariablename>,<adres>,<recordvariablename>)
schrijven  : write(<filevariablename>,<adres>,<recordvariablename>)
```

Als resp. *f*, *a*, *r* de waarden zijn van resp. *filevariablename*, *adres* en *recordvariablename*, dan is het resultaat van de procedure

read resp. *write* dat

van resp. naar het (relatieve) adres *a* in het bestand *f* een record wordt gekopieerd

naar resp. van het recordgebied *r* in het werkgeheugen.

LITERATUURLIJST

- Canning, R. G. Artikelen in diverse nummers van EDP Analyzer: goede kritische artikelen die vooral management-aspecten behandelen.
- Clifton, H. D. Systems Analysis for Business Data Processing, Business Books Ltd. (2nd ed. 1972).
Goed leerboek over klassieke bestandsorganisatie.
- Clifton, H. D. Data Processing Systems Design, Business Books Ltd. (1971).
Dit boek bevat acht uitgebreide cases, goed materiaal.
- Codd, E. F. A Relational Model of Data for Large Shared Data Banks, CACM 13 (1970) 377.
- Database Systems, Infotech Information Ltd (1975).
Lezenswaardige samenvatting van vele aspecten.
- Comer, D. The Ubiquitous B-Tree, Computing Surveys 11 (1979) 121.
- Date, C. J. An Introduction to Database Systems, Addison-Wesley (2nd ed. 1977).
Uitstekend leerboek, met nogal veel nadruk op IMS, en vrijwel niets over fysieke structuren.
- Deen, S. M. Fundamentals of Data Base Systems, Hayden (1977).
Eenvoudige zeer leesbare inleiding.
- Dodd, G. E. Elements of Data Management Systems, Computing Surveys 1 (1969) 117.
Prettig leesbaar overzichtsartikel.

- Knuth, D. E. The Art of Computer Programming, Vol.1 (1974),
Chapter 2: Information Structures, Vol. 3 (1973) -
Sorting and Searching, Addison-Wesley.
Toonaangevende boeken: overigens veel nadruk op
hardware structuren en assemblertaal
- Lefkovitz, D. File Structures for On-line Systems, Spartan Press,
New York (1969).
Wat ouder, maar nog altijd goed boek over dit deel-
gebied.
- Martin, J. Computer Data Base Organization, Prentice-Hall
(2nd ed. 1977).
Goed boek met aandacht voor zowel logische als
fysieke structuren.
- Martin, J. Principles of Data Base Management, Prentice-Hall
(1976).
Eenvoudige uitgave van voorgaande.
- Olle, T. W. The Codasyl Approach to Database Management,
J. Wiley, London (1979).
Hoofdaandacht voor netwerkbenadering.
- Pool, J. A. Optimum Load Factors for Files, IBM, Nederland
van der (1973).
Geeft ook economische beschouwingen over de
directe bestandsorganisatievorm.
- Purchall, F. W. Case Studies in Business Data Processing,
and McMillan (1972).
Walker, R. S. Verzameling goede cases.
- Salton, G. Automatic Information Organization and Retrieval,
McGraw-Hill, New York (1968).
Hoofdaandacht voor documentaire informatie .
- WDBG (Werkgroep Data Base Club van het NGI): Evaluatie
van methoden en technieken voor ontwerp, bouw en
invoering van geautomatiseerde Informatiesystemen,
Den Haag (1980).
- Wedekind, H. Datenorganisation. De Gruyter (1972).
Behandelt de klassieke bestandsorganisatie.
- Wedekind, H. Datenbanksysteme, Bibliographisches Institut,
(1974) Mannheim.

REGISTER

A

achtergrondgegevens 14, 190
adres
-, absoluut 90
-, relatief 90
array 24
attribuut 22

B

balanced tree 62
B-boom 137
bestand 26
bestandsactiviteit 31, 38
bestandsgroei 32
bestandsonderhoud 30, 37, 42
bestandsverandering 31, 39
bestandsverloop 31
bin 96
binaire zoekmethode 86
blok 15
boomstructuur 61
botsing 96
bucket 96
buffer 16
byte 27

C

chaining 108
configuratie 17
consecutieve opslag 32, 36
cylindertabel 123

D

data-base 33, 157
data independence 158

DBMS 159

DDL 159

descriptor 149

direct georganiseerd 32, 90, 137

DML 159

domein 173

E

extraction 98

F

folding 98

funktionele relatie 94

G

gegevens 9

gegevensstructuren 21

geïnverteerd bestand 33, 144

grootvader-vader-zoon 37, 190

H

hashing 96

hiërarchie 122, 167

I

index-direct 135

index-sequentieel 120

informatie 9

informatie-analist 13

informatiebehoefte 11, 12, 47

informatiesysteem, 9, 11

informatieverwerking 9

intern sorteren 77

inverted file 144

invoegen 30, 69
invoegmethode 78

J
journaal 39, 43

K
karakter 27
kettingstructuur 60, 128
knoop 58, 137
kwadratische verschuiving 108

L
lijststructuur 33, 57
lineaire verschuiving 108
lineaire zoekmethode 85
logische structuur 24

M
merge sorting 80, 81
mid-squaring 98
multi-programmering 17
mutatiebestand 30
mutatieproces 44, 50

N
netwerkmodel 170
netwerkstructuur 62
normaliseren 163

O
objekt 21
occurrence 25
onderhoud 30, 37, 109, 133
opslagstructuur 26, 30, 91
overloop 96, 103, 107, 124
overtolligheid 63

P
pad 63
polyphase merge sorting 83
priemgetaldeling 99
primair gebied 96, 103
pijl 58

R
radix conversion 98
random verschuiving 108
record 26
recordstructuur 28
relatie 168
-, directe 92, 97, 111
-, tabel 93, 97, 112, 114
-, funktionele 94, 97
relationeel model 173
repeating group 23
retrieval 33, 144
ringstructuur 60
run
-, balanced 16, 38
-, I/O limited 16

S
samenvoegmethoden 80, 81
scatter storage 107
schema 159
serieel 61
sequentiële bestandsorganisatie 36
set 170
sleutel 26, 90
sleutelconversie 33, 98
sleutelverandering 48
sorteren 76
-, intern 77
-, merge sorting 80, 81
-, externe sortering 81, 84
-, polyphase merge sorting 83
sporentabel 123
stambestand 30
standaardpakketten 13
subschemata 159
synoniemen 95

T
tabel 24
tabelrelatie 93, 97, 112
tak 58
thesaurus 149
trajecttabel 122
truncation 98
type 24

V

vektor 24

veld 27

veranderen 30

verschuiving 108

verwijderen 30

vullingsgraad 103

W

wachttijd 17, 18

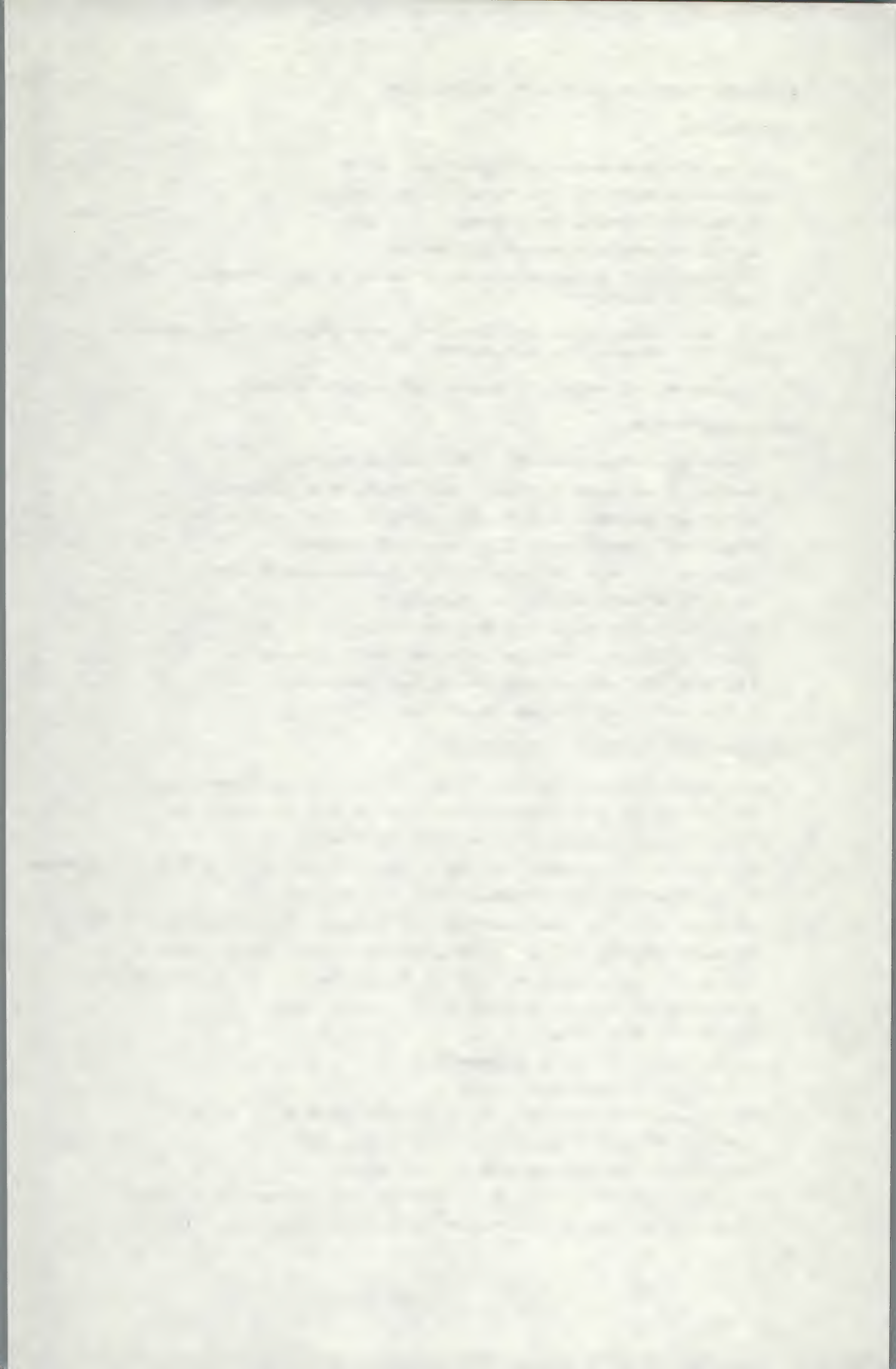
wortel 61

Z

zoekmethode 85, 129

-, binaire 86

-, lineaire 85



ACADEMIC SERVICE INFORMATICA UITGAVEN

INLEIDINGEN

- Computers en onze samenleving* van M.A. Arbib
Basiskennis informatieverwerking van Jan Everink
De viewdata revolutie van S. Fedida en R. Malik
De informatiemaatschappij van Jan Everink
Informatica, een theoretische inleiding van dr. L.P.J. Groenewegen en prof.dr. A. Ollongren
AIV, Automatisering van de informatieverzorging van ir. Th.J. Derksen, drs. H.W. Crins en drs. L.B. Essink
Organisatie, informatie en computers van David M. Kroenke

MICROCOMPUTERS

- Programmeercursus Microsoft BASIC* van Nok van Veen
Werken met bestanden in BASIC van L. Finkel en J.R. Brown
Werken met bestanden in Apple-BASIC van L. Finkel en J.R. Brown
Werken met Visicalc van C. Klitzner en M.J. Plociak, Jr.
CP/M: een gids voor zelfstudie van J.N. Fernandez en R. Ashley
Cursus Z-80 assembleertaal van Roger Hutter
Exidy sorcerer en BASIC van Nok van Veen e.a.
TRS-80 BASIC: een gids voor zelfstudie van B. Albrecht e.a.
TRS-80 BASIC voor gevorderden van Don Inman e.a.
Ontdek de ZX-Spectrum van Tim Hartnell

PROGRAMMEREN/PROGRAMMEERTALEN

- Inleiding tot het programmeren, deel 1* van ir. J.J. van Amstel e.a.
Inleiding tot het programmeren, deel 2 van ir. J.J. van Amstel e.a.
JSP-Jackson structureel programmeren van Henk Jansen
Aspecten van programmeertalen van ir. J.J. van Amstel en ir. J.A.A.M. Poirters
Programmeertalen, een inleiding van ir. J.J. van Amstel e.a.
Het Groot Pascal Spreukenboek van H.F. Ledgerd, P.A. Nagin en J.F. Hueras
Cursus Pascal van prof.dr. A. van der Sluis en drs. C.A.C. Görts
Cursus eenvoudig Pascal van prof.dr. A. van der Sluis en C.A.C. Görts
Inleiding programmeren in Pascal van C. van de Wijngaart
BASIC, EIT-serie, deel 3
Cursus BASIC van ir. R. Bloothoofd e.a.
Cursus COBOL van dr. A. Parkin
Structuur en stijl in COBOL van ir. E. Dürr en dr.ir. F. Mulder
Cursus FORTRAN 77 van J.N.P. Hume en R.C. Holt
Programmeren in LISP van prof.dr. L.L. Steels
Cursus ALGOL 60 van prof.dr. A. van der Sluis en drs. C.A.C. Görts
Programmeren, deel 2: Van analyse tot algoritme van prof.drs. C. Bron

Inleiding programmeren en programmeertechnieken, EIT-serie, deel 1
Inleiding programmeren van prof.dr. R.J. Lunbeck

SYSTEEMPROGRAMMATUUR

Bedrijfssystemen, EIT-serie, deel 4
Systeemprogrammatuur van drs. H. Alblas
Vertalerbouw van drs. H. Alblas e.a.

BESTANDSORGANISATIE/DATABASES

Informatiestructuren, bestandsorganisatie en bestandsontwerp, EIT-serie, deel 5
Gegevensstructuren van R. Engmann e.a.
Bestandsorganisatie van prof.dr. R.J. Lunbeck en drs. F. Remmen
Databases van drs. F. Remmen

INFORMATIEANALYSE/SYSTEEMONTWERP

Vorbereiding van computertoepassingen van prof.dr. A.B. Frielink
Simulatie, een moderne methode van onderzoek van drs. S.K.T. Boersma en ir. T. Hoenderkamp
Systeemontwikkeling volgens S.D.M. van H.B. Eilers
Een samenvatting van de System Development Methodology SDM van PANDATA
Cases op het gebied van administratieve organisatie en informatieverzorging (inclusief systeemontwerp) van prof.dr. P.G. Bosch en H.A. te Rijdt
Uitwerkingenboek bij Cases van prof.dr. P.G. Bosch en H.A. te Rijdt
Gegevensanalyse van R.P. Langerhorst
Evaluation of methods and techniques for the analysis, design and implementation of information systems, editors: J. Blank en M.J. Krijger
Analyse van informatiebehoeften en de inhoudsbeschrijving van een databank van prof.dr. P.G. Bosch en ir. H.M. Heemskerck
Eerlijk en helder van prof.dr. P.G. Bosch

THEORETISCH/COMPUTERSCHAAK/TOEPASSINGEN

Abstracte automaten en grammatica's van prof.dr. A. Ollongren en ir. Th.P. van der Weide
De tekstmachine van dr. M. Boot en drs. H. Koppelaar
Computerschak, schaakwereld en kunstmatige intelligentie van dr. H.J. van den Herik
Lineaire programmering als hulpmiddel bij de besluitvorming van dr. S.W. Douma
Simulatie en sociale systemen, redaktie: J.L.A. Geurts en J.H.L. Oud
Onderneming en overheid in systeem-dynamisch perspectief, redaktie: A.F.G. Hanken en J.H.L. Oud

INFORMATIE OVER DEZE PUBLIKATIES BIJ:

Academic Service, Postbus 96996, 2509 JJ Den Haag
Tel.: 070-247238

